

# NAVAL POSTGRADUATE SCHOOL

## Monterey, California



### THESIS

**INTERNETWORKING:  
AUTOMATED LOCAL AND GLOBAL  
NETWORK MONITORING**

by

Evan B. Edwards  
September, 1996

Thesis Advisor:  
Second Reader:

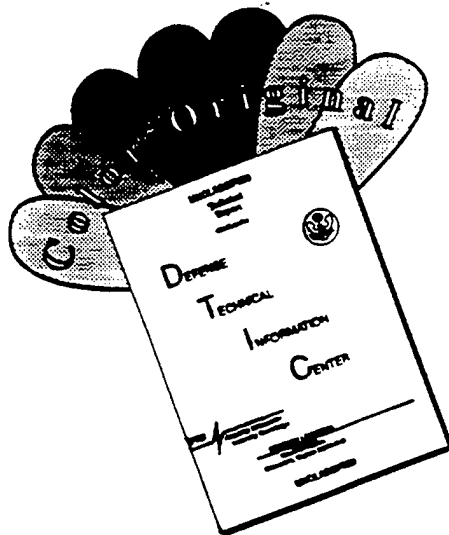
Don Brutzman  
Lou Stevens

19970210 214

**Approved for public release; distribution is unlimited.**

DTIC QUALITY INSPECTED 1

# DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF COLOR PAGES WHICH DO NOT REPRODUCE LEGIBLY ON BLACK AND WHITE MICROFICHE.

<b>REPORT DOCUMENTATION PAGE</b>			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE September 1996		3. REPORT TYPE AND DATES COVERED Master's Thesis
4. TITLE AND SUBTITLE INTERNETWORKING: AUTOMATED LOCAL AND GLOBAL NETWORK MONITORING			5. FUNDING NUMBERS	
6. AUTHOR(S) Edwards, Evan B.				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)			10. SPONSORING / MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution unlimited.			12b. DISTRIBUTION CODE	
13. ABSTRACT. Commercial applications for network monitoring are expensive and therefore not widely available to the majority of network users. Public domain network monitoring software is generally effective in the hands of an expert but difficult to use by the common user because of its command line driven interface. It is a basic tenet of this thesis that network performance and security can be improved if all network users had easy-to-use network monitoring tools available and were encouraged to use them frequently. In this thesis, <i>ping</i> , <i>traceroute</i> , and <i>nslookup</i> were integrated with the familiar user-friendly interface provided by the World Wide Web (WWW) and HyperText Markup Language (HTML) in both automated and interactive versions. These easy to use monitoring tools were evaluated in several working environments at the Naval Postgraduate School and the Monterey Bay Area Network. <i>ping</i> , <i>traceroute</i> and <i>nslookup</i> can now be performed in one-sixth of the time previously required for an expert user. Current network status is now readily available and can be validated at any time through the use of the applications developed in this thesis.				
14. SUBJECT TERMS Network Measurement and Management, Performance Evaluation, Network Security, Local and Global Monitoring Techniques. Common Gateway Interface (CGI), World Wide Web (WWW) Applications, Network Automation. User Interface.			15. NUMBER OF PAGES  261	
			1236. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)  
Prescribed by ANSI Std. Z39-18  
298-102





**Approved for public release; distribution is unlimited.**

**INTERNETWORKING: AUTOMATED LOCAL AND GLOBAL  
NETWORK MONITORING**

Evan B. Edwards  
Lieutenant Commander, United States Navy  
B.S., United States Naval Academy, 1983

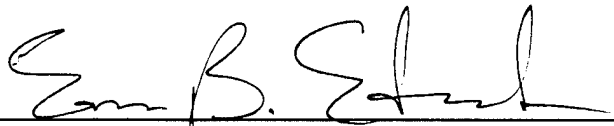
Submitted in partial fulfillment of the requirements for the degree of

**MASTER OF SCIENCE IN COMPUTER SCIENCE**

from the

**NAVAL POSTGRADUATE SCHOOL  
September 1996**

Author:



Evan B. Edwards

Approved by:



Don Brutzman, Thesis Advisor



Lou Stevens, Second Reader



Theodore Lewis, Chair,  
Department of Computer Science



## ABSTRACT

Commercial applications for network monitoring are expensive and therefore not widely available to the majority of network users. Public domain network monitoring software is generally effective in the hands of an expert but difficult to use by the common user because of its command line driven interface. It is a basic tenet of this thesis that network performance and security can be improved if all network users had easy-to-use network monitoring tools available and were encouraged to use them frequently. In this thesis, *ping*, *traceroute*, and *nslookup* were integrated with the familiar user-friendly interface provided by the World Wide Web (WWW) and HyperText Markup Language (HTML) in both automated and interactive versions. These easy to use monitoring tools were evaluated in several working environments at the Naval Postgraduate School and the Monterey Bay Area Network. *ping*, *traceroute* and *nslookup* can now be performed in one-sixth of the time previously required for an expert user. Current network status is now readily available and can be validated at any time through the use of the applications developed in this thesis.



## TABLE OF CONTENTS

I. NETWORK MONITORING FOR LOCAL AND GLOBAL NETWORKS .....	1
A. INTRODUCTION .....	1
B. GOALS.....	3
C. MOTIVATION .....	4
D. METHODOLOGY.....	4
E. THESIS ORGANIZATION .....	5
II. BACKGROUND AND RELATED WORK.....	7
A. INTRODUCTION .....	7
B. BACKGROUND.....	7
C. RELATED WORK .....	8
1. Information Infrastructure Research Group (IIRG) .....	8
2. Internetworking: Multicast and ATM Network Prerequisites for Distance Learning. Murat Tamer.....	9
3. Internetworking: ATM LAN at NPS. Dale Courtney.....	10
4. Internetworking: Using Global ATM Networks for Live Multicast Audio/Video. Ridvan Erdogan.....	11
5. Internetworking: Economical Storage and Retrieval of Digital Audio and Video for Distance Learning. Mike Tiddy.....	12
6. Internetworking: ATM Network Security. Ron Dennis.....	13
7. End-to-End Routing Behavior in the Internet. Vern Paxson.. .....	14
8. Internetworking: Technical Strategy for Implementing the Next Generation Internet Protocol (IPV6) in the Marine Corps Tactical Data Network. Jim Nierle.. .....	14

9. Internetworking: Recommendations on Network Management For K-12 Schools. Dennis Trepanier.....	15
10. Bay Area Gigabit Network (BAGNET).....	16
III. PROBLEM STATEMENT .....	17
A. INTRODUCTION .....	17
B. MOTIVATION .....	17
C. PROBLEM STATEMENT .....	18
D. COMMERCIAL PRODUCTS: NOT THE ANSWER.....	20
E. PROBLEM-SOLVING APPROACH.....	21
F. SUMMARY.....	23
IV. NETWORK MONITORING METHODOLOGY.....	25
A. INTRODUCTION .....	25
B. METHODOLOGY.....	25
C. MONITORING OVERVIEW .....	27
1. <i>ping</i> .....	28
2. <i>ftp</i> .....	32
3. <i>tracert</i> .....	33
4. <i>nslookup</i> .....	34
5. <i>ttcp/nttcp</i> .....	38
6. <i>tcpdump</i> .....	38
D. IP MONITORING .....	39

E. ATM MONITORING .....	40
F. MULTICAST BACKBONE (MBone) MONITORING .....	41
1. <i>mrinfo</i> .....	42
2. <i>map-mbone</i> .....	43
3. <i>mtrace</i> .....	44
G. TECHNIQUES OF NETWORK MONITORING .....	45
1. Cron Daemon Scripts .....	45
2. Automatic Monitoring .....	46
3. Interactive Monitoring .....	47
4. Common Gateway Interface .....	51
5. Human Factors .....	52
6. Topology .....	52
7. Time .....	56
H. CONCLUSION .....	57
V. NETWORK MONITORING TECHNICAL DETAILS .....	59
A. INTRODUCTION .....	59
B. MONITORING CONSIDERATIONS .....	59
C. MONITORING IN THE DEPARTMENT OF DEFENSE .....	61
D. PUBLIC-DOMAIN SOFTWARE TOOLS .....	62
1. <i>ping</i> .....	63
2. <i>ftp</i> .....	64
3. <i>traceroute</i> .....	65

4. <i>ttcp/nttcp</i> .....	67
E. ADDRESS RESOLUTION.....	71
1. IP Addressing.....	72
2. ATM Addressing .....	73
3. Address Resolution Protocol (ARP) Server.....	77
F. NPS MONITORING .....	80
1. Topology .....	80
2. Automated Monitoring.....	83
3. Interactive Monitoring .....	99
G. BAYNET ATM.....	109
1. Interoperability.....	109
2. Incompatibility.....	110
3. Inflexibility and Tariffs.....	110
4. BayNet Monitoring .....	110
5. Future of BayNet.....	111
H. SUMMARY .....	111
VI. SECURITY ASPECTS OF CONTINUOUS NETWORK MONITORING.....	113
A. INTRODUCTION .....	113
B. DESIGN CRITERIA.....	113
C. PURPOSE .....	114
1. Continuous Site Monitoring.....	114
2. Status Warning.....	114



D. METHODOLOGY .....	116
1. NPS Golf Course Lab .....	116
2. Monitoring Algorithms .....	116
E. E-MAIL PAGER NOTIFICATION .....	121
F. AUTOMATIC ARCHIVAL OF MBONE .....	122
G. SUMMARY .....	123
VII. EXPERIMENTAL RESULTS .....	125
A. INTRODUCTION .....	125
B. NPS .....	125
C. AUV 96 .....	126
D. BAYNET .....	127
E. HUMAN INTERFACE .....	129
1. Communication .....	129
2. Authority .....	130
3. Accountability .....	130
F. SUMMARY .....	130
VIII. CONCLUSIONS AND RECOMMENDATIONS .....	133
A. INTRODUCTION .....	133
B. CONCLUSIONS .....	133
C. RECOMMENDATIONS FOR FUTURE WORK .....	136
1. NPS .....	136

2. BayNet.....	137
3. Remote Monitor Management Information Base .....	137
4. Virtual Reality Transport Protocol (VRTP) .....	137
APPENDIX A. DEFINITIONS.....	139
APPENDIX B. <i>ping</i> MAN PAGE (UNIX).....	143
APPENDIX C. <i>ftp</i> MAN PAGE (UNIX) .....	147
APPENDIX D. <i>traceroute</i> MAN PAGE (UNIX) .....	159
APPENDIX E. <i>nslookup</i> MAN PAGE (UNIX).....	163
APPENDIX F. <i>ttcp</i> MAN PAGE (UNIX). .....	169
APPENDIX G. <i>tcpdump</i> MAN PAGE. ....	171
APPENDIX H. <i>mrinfo</i> MAN PAGE .....	189
APPENDIX I. <i>map-mbone</i> MAN PAGE.....	191
APPENDIX J. <i>mtrace</i> MAN PAGE .....	193
APPENDIX K. <i>crontab</i> MAN PAGE (UNIX).....	201
APPENDIX L. <i>rshstl</i> EXECUTABLE.....	203
APPENDIX M. <i>ping_host.pl</i> EXECUTABLE.....	205
APPENDIX N. <i>int_ping.cgi</i> EXECUTABLE.....	211
APPENDIX O. <i>traceroute.cgi</i> EXECUTABLE .....	215
APPENDIX P. <i>nslookup.cgi</i> EXECUTABLE .....	219
APPENDIX Q. MONITORING INSTALLATION INSTRUCTION. ....	223
LIST OF REFERENCES. ....	229
INITIAL DISTRIBUTION LIST. ....	233

## LIST OF FIGURES

4.1. SIMPLE <i>ping</i> .....	49
4.2. <i>ping</i> ACROSS NETWORK BOUNDARIES .....	49
4.3. BayNet PHYSICAL TOPOLOGY .....	55
5.1. LOGICAL TOPOLOGY BETWEEN NPS AND UCSC.....	81
5.2. PHYSICAL TOPOLOGY BETWEEN NPS AND UCSC .....	82
5.3. NETWORK INDEPENDENT DIRECTORY STRUCTURE .....	85
5.4. NPS DIRECTORY STRUCTURE .....	86
5.5. BAYNET IMAGE MAP .....	88
5.6. NPS MONITORING PAGE .....	89
5.7. NPS STATUS .....	92
5.8. NPS <i>ping</i> PAGE.....	100
5.9. COMPLETED INTERACTIVE <i>ping</i> .....	103
5.10. <i>ping</i> RESULTS .....	104
5.11. INTERACTIVE <i>nslookup</i> .....	106
5.12. <i>nslookup</i> results .....	108
6.1. BASIC CONFIGURATION.....	115
6.2. NETWORK INDEPENDENT CONFIGURATION WITH REDUNDANCY .....	117
6.3. NPS CONFIGURATION.....	118
6.4. FAILED <i>ping</i> RESPONSE.....	119
6.5. FAILED <i>ping</i> FROM SUBNET SERVER.....	120
6.6. CONTINGENCY OPERATION.....	121



## ACKNOWLEDGMENTS

It is with sincere heart and deep gratitude that I thank Professor Don Brutzman for being my thesis advisor. I am thankful for his wise council and foresight, and his willingness to share his keen understanding of the Internet way of life. I also thank Lou Stevens for speaking common sense into obscure subject matter. Hearty thanks also to those behind the scenes; Milena Cochran, Don McGregor and the rest of the ever-helpful STL staff.

I wholeheartedly acknowledge the real heroes of this effort: My wife Bonne - I could not have done this without her by my side, and my children, Jesse (11), Luke (9), Nina (5), Olivia (4), Hope and Harry (both 2).

Last but really most important, I thank God who holds the keys to learning. I do all things by His outstretched arm.



## **I. NETWORK MONITORING FOR LOCAL AND GLOBAL NETWORKS**

### **A. INTRODUCTION**

The key problem which this thesis addresses is the lack of capability to monitor networks, particularly large networks and internetworks. Network monitoring is hard and it's difficulty increases with size. Regardless of network size however, monitoring must be addressed. Obstacles to this end are many. The exact number of users is rarely known. There is no such thing as a typical connection or a typical application. Topology changes frequently. Exponential growth makes internetwork monitoring a moving target. To monitor a network means to have the capacity to determine the route taken by the transmission, display the time that it required, and determine what percentage (if any), of the transmission was lost. Monitoring also includes the ability to diagnose when, where, why and how problems occur.

Network monitoring affects every network user. Remarkably, monitoring deficiencies are still readily apparent in most networks. Not only is the monitoring problem difficult in and of itself, but there exists a lack of user-friendly easy-to-implement tools to attack it. Commercial vendors promote various wares as the answer. Unfortunately the same situation exists after the costly addition of commercial monitoring platforms: few people actually understand what is going on with their network.

The Department of Defense (DoD) prides itself on professionalism at all levels. Aviators don't just learn to fly, they also learn the theory of aerodynamics. Jet mechanics don't just learn how to repair jet engines they also learn the theory of propulsion. Achievement of their respective immediate tasks may not always depend on their knowledge of the theory. In the long

run however, their performance is much greater than poorly trained counterparts in some other countries. Why is it that network operators and users attempt to manage and utilize costly, advanced networks without a solitary clue about how things actually work?

The prevailing attitude is that network monitoring is fundamentally beyond the grasp of most network users. It is true that network monitoring is not an easy problem. It is also true that users across the board have little understanding of the basics of network behavior. This is the current status quo of network monitoring but it is not an acceptable end.

Turning a blind eye to monitoring enslaves network users and administrators alike. Users are without any understanding of what is actually controlling and affecting their applications. Network administrators are chained to a network which no one else can understand, assuming that the administrator understands it. Administrative personnel are the only ones who can troubleshoot and correct all but the most mundane networking problems.

Performance suffers for a number of reasons. How and why the performance degradation occurs is a tough problem. The amount of performance loss is again difficult to measure. As networks continue to grow and the applications that run on them get more complex, performance evaluation is more critical than ever before. Accurate performance measurement and analysis must be given priority.

The solution to the monitoring problem is to give users an understanding of the basics of network monitoring, the ability to perform basic network monitoring tasks simply and efficiently and the ability to evaluate their networking environment. Users and administrators alike can greatly benefit from a genuine understanding of their network and their connection to the Internet.



To allay monitoring illiteracy certain steps are required. The first task is to put network monitoring tools in the hands of the users. These tools are the many public-domain software applications that fully accomplish desired monitoring ends. They are extremely underutilized because they are operated via cryptic command line interaction. The second task is to implement a monitoring solution without significant hardware or software investment. Commercial proprietary systems are expensive to purchase, support and operate. Public-domain software is proven in most cases and requires no additional hardware. Third, the interface to the monitoring capability needs to be portable, extensible and user-friendly. HTML, Common Gateway Interface (CGI), *perl* scripts and the World Wide Web provide these capabilities.

## **B. GOALS**

The purpose of this thesis is to evaluate and integrate existing public-domain network monitoring tools. The initial arena to test the monitoring tools is the Naval Postgraduate School (NPS) System Technology Lab (STL) Local-Area Network (LAN). Following successful development of these tools in the local environment, they will be tested on the Monterey Bay Area Network (BayNet) using both IP and ATM backbones. This will enhance BayNet collaboration not just in monitoring aspects but in other applications as well. A definition of terms and abbreviations is included as Appendix A.

## **C. MOTIVATION**

Many network users at NPS have little understanding of the campus internetwork. BayNet suffers the same problem. Understanding local and regional networks is important to internetwork health. This understanding must not be limited to network super-users. A tremendous amount of information is available to all users. Current network status, server status, topology and network addressing are things that users need to know. Unfortunately, few users know how to obtain this data.

Understanding networking at NPS requires knowledge of Internet Protocol (IP) and a grasp of how NPS fits into the regional internetwork. Additionally, the NPS ATM LAN which is currently in a developmental phase exists side-by-side with the campus IP network. So to genuinely understand NPS networking requires knowledge of the BayNet ATM network. After a thorough understanding of the local and regional networks is attained, local and regional network behavior will become familiar and true monitoring can begin. NPS plans to participate in a wide range of Internet-compatible traffic from electronic mail (e-mail) to complex real-time voice, video and graphic applications, both over IP and ATM. The only way to ensure that these applications mature in a robust and secure environment is through diligent monitoring efforts.

## **D. METHODOLOGY**

Network monitoring and performance evaluation are non-existent on most networks. Monitoring success can mean virtually any progress that lays open previously hidden network behavior. There are however certain principles and practices which will help provide meaningful and useful monitoring. The hosts upon which the network rely must be tested on a regular basis.

Problems need to be reported and documented as they occur. The ability to test network hosts interactively must be available but the user must be shielded from the obscure complexities of the command line environment. To accomplish this a user-friendly interface to network monitoring is required. Automation of monitoring tools speeds the entire monitoring and trouble-shooting process. Software tools that can aid these monitoring endeavors will be automated with scripting and simplified user interfaces. Finally network performance must be measured and analyzed. This work develops and evaluates software using all of these techniques.

## **E. THESIS ORGANIZATION**

Chapter II. Background And Related Work.

Chapter III. Problem Statement.

Chapter IV. Methodology.

Chapter V. Network Monitoring Technical Details.

Chapter VI. Security Aspects of Network monitoring.

Chapter VII. Experimental Results.

Chapter VIII. Conclusions and Recommendations.



## **II. BACKGROUND AND RELATED WORK**

### **A. INTRODUCTION**

This thesis is part of an ongoing research effort at the Naval Postgraduate School. This endeavor involves participants comprised of staff and students at NPS, California State University Monterey Bay (CSUMB), University of California Santa Cruz (UCSC), Monterey Bay Aquarium, Monterey Bay Aquarium Research Institute (MBARI), Moss Landing Marine Landing (MLML) and numerous national and global partners. The research efforts revolve around standard Internet connections and high-bandwidth, low-latency network applications. Areas of interest include Multicast Backbone (MBone), Asynchronous Transfer Mode (ATM), wireless networking, Video Tele-Conferencing (VTC), and IP Version 6 (IPV6) among others. This broad approach to network issues affords the greatest potential of meeting the future of networking head-on.

### **B. BACKGROUND**

The key problem which this thesis addresses is the inability to accurately and efficiently monitor networks, particularly large networks and internetworks. To monitor a network or internetwork means to have the capacity to determine the route taken by the transmission, resolve the time that the transmission required, and determine what percentage (if any), of the transmission was lost. It includes the ability to determine when, where and why problems occur. Monitoring deficiencies are more apparent as network size increases. Regardless of network size however, proper monitoring is

crucial. Trial-and-error network monitoring and management have sufficed until recently because networks were small. As networks have grown in both size and popularity, the complexity of the monitoring problem has increased. On those networks which possess monitoring capabilities, the information is not readily available or easily interpreted. The responsibility to monitor a network therefore falls solely on the system administrator. Most network administrators are already overworked due to numerous challenging system duties, so network monitoring is often deferred or avoided. The purpose of this thesis is to develop tools that allow network administrators and users to accurately and easily monitor network performance.

The NPS System Technology Lab (STL) IP and ATM LANs are the networks on which initial monitoring tools will be tested. Following successful development of these tools in the local environment, they will be tested on the Monterey Bay Network (BayNet). Testing on a national level will follow BayNet evaluation. This will enhance collaboration with partners at Information Wide-Area Year (IWAY), Old Dominion University (ODU), Naval Undersea Warfare Center (NWC), Newport, RI and others for network monitoring and other applications as well.

## **C. RELATED WORK**

### **1. Information Infrastructure Research Group (IIRG)**

The IIRG is a multidisciplinary group of NPS students and staff led by Dr. Don Brutzman. The objectives of IIRG are to connect everyone to everything, stimulate imaginative and "out-of-the-box" thinking by hands-on experience in real-world

situations, overcome technical constraints through technical solutions, and solve people problems with people solutions. The primary locus of IIRG pursuits is the proper use of globally shared resources on the Internet. IIRG endeavors include involvement in the following projects: internetworking K-12 schools and institutions, research in large-scale virtual environments (LSVEs), Asynchronous Transfer Mode (ATM) internetworking and applications, Multicast Backbone (MBone) implementations and enhancements, IP over seawater (IP/SW), and projects which internetwork U.S. Navy. Brief descriptions of IIRG work related to this thesis follow.

## **2. Internetworking: Multicast and ATM Network Prerequisites for Distance Learning. Murat Tamer.**

The Internet, the World Wide Web and the Multicast Backbone (MBone) have been used in a variety of ways for distance learning. VTC classrooms have obvious value and utility but they are limited to communicate with only a small number of similar VTC facilities. We are most interested in open solutions which take advantage of the global Internet. Therefore the problem addressed by this thesis is to evaluate the specific benefits and drawbacks of Internet technologies in support of distance learning.

This thesis includes a detailed examination of MBone, Asynchronous Transfer Mode (ATM) and the Distributed Interactive Simulation (DIS) protocol from the perspective of distance learning. An innovative design for a low-cost Web/MBone-capable classroom is presented. Experimental results include globally multicasting the

IEEE Autonomous Underwater Vehicle (AUV 96) conference and digitally recording the 1996 Monterey Bay Web Content and Access Workshop.

One result we found is that MBone can be used successfully for distance learning purposes despite common constraints of limited (128 Kbps) bandwidth. A further result is that an MBone classroom can be 42% as expensive as a VTC classroom if an SGI Indy is used and 12% as expensive as a VTC classroom if a PC is used in the classroom. Consequently many schools can afford Internet-based distance learning using the solutions presented in this thesis even though they cannot afford VTC rooms.

### **3. Internetworking: ATM LAN at NPS. Dale Courtney.**

ATM cell relay is a high-speed, packet-switched transport service designed for unifying diverse traffic - telephony, data, audio, video, and image - over common facilities in fixed length cells. ATM is well-suited for real time, delay-sensitive, broadband applications such as multimedia, real-time workgroup computing, imaging, and live audio/video. ATM provides a solution that can work seamlessly between LAN, regional WAN, and global internetworks.

This thesis is being written in direct response to the Pacific Bell (PacBell) California Research Network (CalReN) \$25 million program to stimulate the development of new applications for high-speed data communications services. CalReN application development work established a foundation for broadband services including on-line schools, distance learning, and Internet-based curriculum development and delivery.



The objective of this project is to create, test, and build an electronic information infrastructure at NPS based on ATM cell relay that will provide for tele-education, telescience experiments, and digital interactive multimedia. This thesis lays the groundwork for future ATM work by sending real-time audio, video, and graphics.

The end of the CalReN grant poses a significant barrier to further NPS off-campus ATM research. External ATM connectivity will likely cease given the cost for ATM network access.

Hardware interoperability problems are another major barrier to continued NPS ATM research. NPS operates Cisco switches and Fore NICs. The school cannot take full advantage of the proprietary benefits of either company and must set all configurations manually. UCSC uses proprietary Parallax hardware. They and other BayNet participants are able to achieve limited but satisfactory functionality.

Lastly, the Cisco A-100 switches that NPS purchased are limited to 155 Mbps. This restricts the school's ability to explore the Gbps barrier, requiring investing in new switches across the campus when this technology becomes feasible.

#### **4. Internetworking: Using Global ATM Networks for Live Multicast Audio/Video. Ridvan Erdogan.**

Monterey BayNet is a regional wide-area network (WAN) which connects K-12 schools, libraries, research institutions and institutions of higher education throughout the Monterey Bay Area. Its purpose is to increase the quality of education in this region. The BayNet K-12 project enables students and teachers to efficiently access a wide range of information and resources previously available only via hard copy and at great expense.

Distance learning has a positive impact on the quality of education and training. It provides information to people when and where they need it. Current distance learning technologies, such as video teleconferencing, are not scaleable and are prohibitively expensive. Distance learning via the Internet, particularly through Multicast Backbone applications is an economically feasible approach with a proven record.

The MBone solution is available to Monterey BayNet but a distinct lack of interaction exists. BayNet sites have full access to the Internet yet they are not able to take advantage of the functionality provided by the MBone.

This thesis documents the implementation of MBone over Monterey BayNet for educational purposes. It enumerates the changes required to re-configure BayNet sites for MBone connectivity. It demonstrates that MBone over Frame Relay is possible and that the current MBone technology provides excellent performance even on slower networks.

## **5. Internetworking: Economical Storage and Retrieval of Digital Audio and Video for Distance Learning. Mike Tiddy.**

This thesis focuses on the testing and comparison of currently available methods of digital audio and video (A/V) storage and the use of current transfer modes and protocols for on-demand retrieval of A/V over the Internet. Different compression methods, file formats, and World Wide Web applications will be examined. No new compression techniques are developed, but rather existing standards are being researched.

This thesis follows the "Recommendations for Future Work" section of an NPS thesis written by Tracey L. Emswiler. The MBone provides the ability to send and receive near-real-time A/V over the Internet but currently provides no way to efficiently

store the data and replay the session on demand. Tools that allow a user to play audio and video that is retrieved over the Internet do exist but force the user to transfer the entire file before it is played. The file sizes that accompany a 30-minute video are so great (about 1 GB) that this method of retrieval is not a reasonable alternative due to data transmission speeds and user storage space limitations. Thus, practical compression and practical network distribution are the key bottlenecks.

#### **6. Internetworking: Network Security. Ron Dennis.**

Originally the exclusive domain of scientists and researchers, the Internet is used today by an estimated twenty million users around the world. File transfer, data base access, bulletin board access, distance learning, and financial transactions comprise the majority of use today. Security incidents grows proportionately with the number of Internet users

Many believe that new network technologies such as Asynchronous Transfer Mode (ATM) will significantly reduce the threats to network security. Regardless of the technology and protocols used for the Internet backbone current networking technology will remain in place for the near future. Additionally, the internetworking trend will continue, and with it, the need for a focus on computer and network security. Personal data, credit card numbers, proprietary business information, and sensitive research data must be protected both while it is stored in host systems and while being transmitted over the network. Unlike other narrowly focused fields of Computer Science, computer and network security is simultaneously a technical, social, and managerial discipline. This

thesis focuses on the vulnerabilities associated with internetworking a combined Internet Protocol (IP) Ethernet and Asynchronous Transfer Mode (ATM) local-area network at the Naval Postgraduate School over a wide area.

#### **7. End-to-End Routing Behavior in the Internet. Vern Paxson.**

The problem which this work addresses is that the behavior of the Internet is basically unknown. No one knows how many users there are although the number of hosts is periodically measured. There is no typical behavior pattern. There is no dominant application. Performance figures are equally lacking. No one knows the number of packets dropped or queued. Finally, Internet simulation is difficult due in large part to the self-similar behavior modeling deficiencies.

This work reports on 40,000 end-to-end traceroute measurements over 37 Internet sites. Data was collected on pathology conditions, routing stability, routing loops, erroneous routing, and routing symmetry. Further research involves continuous documentation at a greater number of sites using specially designed software. This data will provide an accurate base from which to determine Internet behavior and build accurate simulations.

#### **8. Internetworking: Technical Strategy for Implementing the Next Generation Internet Protocol (IPV6) in the Marine Corps Tactical Data Network. Jim Nierle.**

The Marine Corps must architect a tactical internet based on a software technology that is in transition - the Internet Protocol (IP). Development of the Marine Corps' Tactical Internetworking System (Tactical Data Network or TDN) is progressing

concurrently with the global Internet community's development of the Next Generation Internet Protocol (IPV6). Current (IPV4) and next generation (IPV6) versions of the Internet Protocol can together meet the tactical internetworking needs of the Marine Corps.

IPV4 provides universal interoperability with other networking technologies and support for a wide range of services now, but without enhancements IPV4 cannot meet the long-term needs of evolving tactical applications. IPV6 is needed to meet emerging requirements (such as secure mobility) but is not yet ready for implementation in the Tactical Data Network. Therefore the Marine Corps must build the tactical internet architecture using IPV4 and incorporate IPV6 improvements when transition is possible.

Marine Corps commitment to IP is essential to ensure universal interoperability and hardware-independent evolution of tactical applications and networking technology. This work presents a tactical IP addressing plan for TDN that works with IPV4 and also facilitates smooth transition to IPV6. In concert with the other military services, the Marine Corps must develop a strategy for migrating the joint tactical internet to IPV6. The future viability of the Tactical Data Network depends on the Internet Protocol.

#### **9. Internetworking: Recommendations on Network Management For K-12 Schools. Dennis Trepanier.**

The introduction of the Internet into the K-12 educational environment presents numerous challenges. At the same time, it represents a tremendous opportunity to enrich the classroom. Curricula shortages can be overcome, new techniques of learning can be applied, teacher-to-student ratio problems diminish, and technology can be introduced

early rather than after an entire publishing cycle has passed. The challenges to networking in the classroom addressed in this work are the introduction to and the management of network assets in the classroom. Introduction of network learning into the classroom includes providing network service advice, recommending software decisions and assisting in the installation of tools, and determining the optimal mix of new technology and traditional studies.

## **10. BAGNET**

BAGNet is the first and largest of the CalREN projects and is the outgrowth of a several year effort to establish an ATM test bed in the San Francisco Bay area. The project participants include most of the academic, government, and industry computer science research organizations in the Bay area.

The goals of the project are to establish an ATM infrastructure, to engage all of the participants in an application that was uniquely suited to a high speed, metropolitan area ATM network, and to enable several diverse applications that involved two or three of the participants directly collaborating. BAGNet is currently inactive.

### **III. PROBLEM STATEMENT**

#### **A. INTRODUCTION**

Network monitoring is a vast frontier. In the two decades from 1966 to 1987, there were several thousand papers published on queuing but just forty-four on traffic measurement. [Paxson, 96] It is a tricky subject. Networks are growing exponentially and bandwidth is a limited commodity. Monitoring must come to the forefront of the network discipline. The networking community must police itself or more drastic measures will undoubtedly be taken.

Commercial products are available which perform some or all of the necessary functions required to adequately monitor a network. They are not the right answer. The commercial approach is expensive, often proprietary and are targeted for administrators.

#### **B. MOTIVATION**

Networks have become an integral part of all aspects of Department of Defense (DoD) operations. The explosion of networking has changed the face of not just the military but also business, education, medicine, industry, finance and government. The ability to monitor and control network behavior is critical. Monitoring is a crucial part of proper network administration. In the DoD, mission accomplishment (and hence life itself) may hinge on network performance. Network monitoring is also a key component of network security.

The advance in network technology is unbelievable. Network providers claim mind-boggling performance and astonishing reliability. Statisticians call attention to the phenomenal growth of the Internet. In reality, reliable and accurate performance figures are few and far between. No one really knows how many terminants are on the Internet. [Paxson, 96] Network monitoring has made little progress while networks themselves are growing in size and capability. Both the DoD and the civilian sector are in need of solutions to this complex monitoring problem.

### **C. PROBLEM STATEMENT**

Networks are difficult to monitor. To effectively monitor a network means to fully understand the behavior of all traffic into and out of the network. Numerous variables exist in this pursuit. These differences include protocol variations, hardware interoperability concerns, incompatible operating systems, bandwidth allocation issues, security requirements and address resolution and domain constraints, to name a few.

Because of the global reliance on internetworking, proper monitoring is more important than ever before. During its early years, the Internet was used almost exclusively by the academic and research disciplines. Now its importance is far flung. Internetworking is absolutely crucial to the finance and business industries. Military and industry communications can be severely crippled without Internet access. In this arena, effective network diagnosis and control through monitoring and evaluation must be exercised.



Performance evaluation is particularly elusive. Proper network performance evaluation provides real-time knowledge of traffic arrival and delivery statistics including transfer rate, cell loss rate, and route taken. Performance evaluation is absolutely essential to the proper utilization of a network. It is difficult because the layered network protocol is meant to shield the user from low-level considerations. While this transparency is essential, it has the unfortunate side-effect of turning performance measurement into a black science.

Performance lies at the center of a number of highly visible and consequential issues. These include actual versus delivered performance, quality-of-service guarantees, and bandwidth allocation. Performance figures can be obtained in numerous ways, each subtly presenting their own brand of errors. Obviously if given a choice, performance figures will be portrayed which support the case of the presenter. Only through a painstaking effort to ensure validity will accurate figures come forth.

The current laissez-faire approach to network monitoring is totally unsatisfactory. Because of both the widespread dependence on internetworking and the sophistication of modern network applications, effective monitoring is essential. Monitoring ability and responsibility must evolve into an inherent part of every network but at the same time, they must not adversely impact network performance or security.

Public-domain software tools exist that accomplish some or all of these goals. That is the good news. The bad news is that they are command-line driven and may require root super-user permission. Commercial tools are also available but suffer two

great disadvantages: prohibitive cost and proprietary nature. These disadvantages are abhorrent in an age when resources are slight and interoperability is key.

Monitoring tools are needed that are effective and robust. They must be user-friendly and system-friendly. This means they have to be easy to use, easy to access, easy to understand, easy on data storage and easy on system performance. They must do all of this completely transparently to the user, providing needed information in an understandable way.

#### **D. COMMERCIAL PRODUCTS: NOT THE ANSWER**

There are commercial network monitoring solutions available. They possess advantages and disadvantages. The advantage of a commercial system is that represents a centralized, visible interface to network monitoring and performance tracking. The disadvantages are many.

Commercial solutions may be comprised of software, hardware or both. Below are some example features from a low-cost software-only monitoring package which sells for over 500 dollars.

- assignment of alphanumeric user names to network addresses for designating stations by familiar names rather than physical addresses.
- Generates network traffic for troubleshooting, capacity planning and stress testing.
- Monitors network activity of all communicating station pairs in real-time.
- Performs a full 7-layer, color coded, decode of frames which have been captured. [Triticom, 95]

The same testing can be accomplished using public domain software and (with a limited effort), the results creatively displayed. A representative combined hardware and software solution to ATM network monitoring costs over 10,000 dollars. The system provides collection of cell-level traffic. [Attila, 95]

If one thing is certain in networking, things are going to change. The protocol of today may be gone tomorrow. Architectures, operating systems and interfaces may fall from grace as quickly as the companies that designed and implemented them. Proprietary solutions are not a wise choice even though the service that they provide is.

Free, globally available monitoring software is important especially for non-profit organizations such as the DoD and academic institutions. While the monitoring of their networks is no less important than the civilian sector and in certain cases obviously more so, such organizations do not have the ability to support resource dependent monitoring solutions. Few non-profit institutions can afford to spend 10,000 dollars per unit for a monitoring device. The cost of monitoring must be upheld by the networking community as a whole.

## **E. PROBLEM-SOLVING APPROACH**

The network monitoring tools designed in this thesis are meant to be beneficial to all network users. They are designed around HTML pages and CGI scripts. The design methodology follows a stepping-stone approach. The first step is to develop a familiarity with the public-domain applications and note their behavior on local networks. Next

design, test and implement an automated status checking system for NPS campus networks. An inherent capability of the automated monitoring is self-documentation. Any host "down" status will be documented and archived. After the automatic monitoring is in place and functional, an interactive monitoring capability will be developed. Interactive monitoring makes common network tools such as *ping*, *traceroute* and *nslookup* available through the use of a Web browser rather than command line interaction. After the NPS automated and interactive tools have been solidified, a central monitoring location will be developed. This URL will be a single destination available to all network users. The site will house a wealth of information regarding the status of the network, operating system considerations, phone numbers and any other administrative data that may be useful in a troubleshooting endeavor. The next step will be to gain the ability to notify appropriate personnel in the event of network problems. This will occur via e-mail. After the full monitoring application has been adequately tested at NPS, it will be ported to BayNet. BayNet is currently unmonitored for the most part. These tools will greatly enhance the ability to determine causes of network slowdowns and lost connections. A brief synopsis of the approach follows:

1. automate status verification.
2. self-document.
3. interactive monitoring.
4. centrally located monitoring location (URL).
5. automatic monitoring failure notification.
6. port application to BayNet.

## **F. SUMMARY**

Network monitoring requires attention. It is virtually non-existent for most users on many networks. Global reliance on internetworking dictates that this situation change. Without effective monitoring, trouble-shooting efforts are trial-and-error and network performance suffers. There are many applications available to enable network monitoring without a large expenditure for either monitoring hardware or software.



## IV. NETWORK MONITORING METHODOLOGY

### A. INTRODUCTION

This chapter covers the basic building blocks of effective network monitoring. Public-domain tools that enable network status-checking, topology and performance measurement are discussed briefly with their main advantages and disadvantages. IP, ATM and MBone specific behavior and related pitfalls follow. The techniques of network monitoring are covered next. These include system activated *perl* scripts for automated monitoring and CGI interactive monitoring tools for real-time feedback. Human factors are also illustrated as the common denominator of network decision making even though the most forgotten. Understanding topology is indispensable whether on a local or global scale. Physical and Logical topologies of both the NPS campus network and Bay Area Network are covered in detail. These topologies are a significant factor in the design and implementation of the tools discussed in this thesis.

### B. METHODOLOGY

Monitoring can be either active or passive. Active monitoring is accomplished through a program specifically designed to test and analyze a network. Passive monitoring merely analyzed the performance of the existing applications. [Paxson, 96] In either case, effective monitoring can be accomplished using software that is readily available. The main problem is that most networks are not monitored at all, or only in a crisis. The correct approach is to implement a proficient and dependable monitoring suite that operates

continuously. The suite is comprised of static and interactive monitoring capabilities, easy-to-use interfaces, and self-sustaining operation. Network information such as host status, points-of-contact, routing information and topology must be current and easily obtainable.

The ideal solution is to provide “one-stop shopping” for network monitoring. Everything that is needed to gain a full and current estimation of network status located in one place. The output results can be granularity-layered to eliminate user saturation. Having network vital statistics readily available and elegantly displayed makes troubleshooting and diagnosis an efficient and effective endeavor. Network problem solving is thus transformed from a frustrating exercise (immediately followed by a call for the harried system administrator) to a visit to a URL and some semblance of problem resolution.

One of the primary components of monitoring is automatic host verification. Automated system testing of designated network hosts can occur on a regular basis and the results thereof made available via a HyperText Markup Language (HTML) page. In the event of host failure, appropriate personnel need to be immediately (and automatically) notified. Automated monitoring makes the network self-monitoring and self-documenting.

The presence of automated monitoring does not eliminate the need for interactive real-time monitoring. When network behavior is suspect, a fast and comprehensible user interface to the common monitoring tools is an excellent way to quickly isolate the problem. The tools themselves and their common advantages and disadvantages are covered in following sections.

Finally, a well-structured archive of historic system performance is a means of documenting and demonstrating the performance of the network over a given period of time.



The storage of such archives can be accomplished through easily interpreted text files. This method of archival requires minimal storage and can be fully automated. If properly configured, interpretation of the archives is self-evident. Additional automated analysis of well-structured archives remains a promising area for future work.

### **C. MONITORING OVERVIEW**

Networks come in many shapes and sizes. Although they function differently, they share numerous characteristics. The wide range of network scenarios which result from shared versus switched media, server configuration differences and operating system versions do not significantly alter the theory of network monitoring. Just as cars have four tires and a motor, networks have nodes and connections. The start of network evaluation is to ensure that the nodes are talking to each other. The common tests work in similar fashion on most networks, and small adjustments usually will enable them to operate as expected even in difficult hardware and software configurations.

At the heart of network monitoring is the ability to test whether a number of hosts on a network are operational. All other monitoring data must be built on this foundation. Host status most often depends on the status of other hosts such as the web server, the domain name server and the Address Resolution Protocol (ARP) server. It is therefore necessary to know which hosts perform these functions. The ability to ascertain the route that a packet takes is important. The performance of the packet route is also important. Address resolution, address assignment, routing and signaling protocols and their associated failure

modes must be thoroughly understood once connectivity is established. Performance evaluation for our purposes means throughput measured in Megabits per second (Mbps).

There are many tools available to accomplish these tasks but many users don't know how to use them. A study was conducted at Lawrence Berkeley National Laboratory that revealed that 40 percent of file transfer protocol (*ftp*) queries into their site were failures. [Paxson, 96] They failed because of user error rather than system problems. The public domain applications are widely documented but suffer a few handicaps. They are executed from the command line. They are often add-ons to operating systems and as such are not standardized. Results are often difficult to understand. Limits on the tools may be imposed by system personnel due to the possible adverse affect these tools may have on network performance if used improperly. Use of the tools may even be limited to system administrators exclusively. [Cochran, 96] Training in their effective use is often not available. When properly used, however, these public-domain software applications provide all that is necessary to diagnose network behavior. A brief description of common network tools follows. Examining capabilities and pitfalls is an essential prerequisite to building Web-accessible automated monitoring scripts which utilize these public-domain tools.

### **1. *ping***

*ping* is a tool for network testing, measurement and management. It utilizes the ICMP protocol's ECHO\_REQUEST datagram to elicit an ICMP ECHO\_RESPONSE from a host or gateway. ECHO\_REQUEST datagrams ("*pings*") have an IP and ICMP header, followed by an 8-byte timestamp, and then an arbitrary number of 'pad' bytes used to fill out

the packet. The *ping* man page is included as Appendix B. UNIX man pages are included to give the reader a thorough understanding of the technical aspects of the applications.

*ping* is the most common network monitoring tool. *ping*'s strength is that it allows the user to tell immediately whether a host is 'alive' or not. Disadvantages of *ping* are that the user must know the proper name or possibly even the IP address of the host in question. The user needs to be familiar with the configuration of the *ping* command on the initiating host because *ping* command line specifications and results vary significantly. This usually means that the user must view the man page for their local operation system to determine which switches to set at the command line. Negative feedback is vague. The types of usage fits generally into two categories, verbose and terse. This first example is a test of the terse form.

```
% ping navy.stl.nps.navy.mil
navy.stl.nps.navy.mil is alive
```

This next example is the same test with a small error in the input.

```
% ping royal.nps.navy.mil
ping: unknown host royal.nps.navy.mil
```

Upon examination of the result, the user types the correct domain name syntax to get the correct result.

```
% ping royal.stl.nps.navy.mil
royal.stl.nps.navy.mil is alive
```

The terse *ping* is a streamlined test to check the immediate status of the host in question. It is assumed that the user is familiar with the host, that the domain name service is properly configured and able to find the host, and that further information is not required.

The following *ping* yields a verbose result. This test will run continuously until stopped by ^C (control C).

```
% ping azure
PING azure.stl.nps.navy.mil (131.120.64.4): 56 data bytes
64 bytes from 131.120.64.4: icmp_seq=0 ttl=255 time=1 ms
64 bytes from 131.120.64.4: icmp_seq=1 ttl=255 time=1 ms
64 bytes from 131.120.64.4: icmp_seq=2 ttl=255 time=1 ms
64 bytes from 131.120.64.4: icmp_seq=3 ttl=255 time=1 ms
64 bytes from 131.120.64.4: icmp_seq=4 ttl=255 time=1 ms
64 bytes from 131.120.64.4: icmp_seq=5 ttl=255 time=1 ms
^C
----azure.stl.nps.navy.mil PING Statistics----
6 packets transmitted, 6 packets received, 0% packet loss
round-trip min/avg/max = 1/1/1 ms
```

It is significant that this *ping* must be stopped by using ^C. For the user unfamiliar with the test, there is no indication of how to stop the test. This may lead to more drastic steps such as turning off the power to the unit or just letting the *ping* go on indefinitely, further degrading network performance.

The next test uses the verbose format but with the addition of a switch to set the number of *pings* sent to one (1). This frees the user from having to terminate the *ping* by typing ^C.

```
% ping -c 1 azure
PING azure.stl.nps.navy.mil (131.120.64.4): 56 data bytes
64 bytes from 131.120.64.4: icmp_seq=0 ttl=255 time=1 ms

----azure.stl.nps.navy.mil PING Statistics----
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 1/1/1 ms
```

The ability to determine the number of packets sent is beneficial. Not only does it lessen error potential, it alleviates the guesswork. However, the syntax of the switch can vary from host to host. This is the same *ping* as above on a host that uses alternate syntax.

```
% ping -c 1 royal
ping: illegal option -- c
usage: ping host [timeout]
usage: ping -s[drvRlLn] [-I interval] [-t ttl] [-i interface]
        host [data size] [npackets]
```

*ping* is an essential part of any network monitoring suite. It suffers from the disadvantages of platform-dependent functionality, inconsistent format and poor feedback. Furthermore *ping* is capable of damage. If used improperly, it can saturate a network. Users

must be able to use the man page and be aware of the damage that *ping* can do if used incorrectly.

## **2. *ftp***

*ftp* is the user interface to the Internet-standard File Transfer Protocol. The program allows a user to transfer files to and from a remote network site. It is not primarily a monitoring tool but it can be used for that purpose. The *ftp* man page is included as Appendix C.

*ftp* is useful for determining a rough measure of network performance. There is no feedback from *ftp* until it has completed the transfer. When it has completed, transfer statistics are displayed with an estimate of the transfer rate. Below is a transfer of a 2.5 MB uncompressed binary. During the portion labeled [delay], 22 seconds pass with no user feedback.

```
% ftp cs.nps.navy.mil
Connected to cs.nps.navy.mil.
220 taurusftp server
(Version 6.10 Wed Mar 18 11:57:03 PST 1992) ready.
Name (cs:edwardse):
331 Password required for edwardse.
Password:*****
230 User edwardse logged in.
ftp> get n16e201.exe
200 PORT command successful.
150 Opening ASCII mode data connection for
n16e201.exe (2507571 bytes).
```

[delay]

226 Transfer complete.

local: n16e201.exe remote: n16e201.exe

2517841 bytes received in 22 seconds (1.1e+02 Kbytes/s)

ftp> quit

The transfer rate given at *ftp* completion is not accurate enough for performance analysis because it is only a partial measurement of the packet flow across the physical route(s). Other factors are present that can definitely impact the transfer rate given in *ftp*. Network traffic flow varies considerably. Bursts of data can occur at rates far from the average transfer rate. Transfers of small files do not allow enough time to flatten the effect of bursty behavior. Simply increasing the size of the file will not ameliorate the inaccuracies of *ftp* transfer rate data. Network and end host load is not constant. Normal office hours, popular dial-up access times and network 'witching hours' are predictably high use periods as well. Nonetheless, *ftp* can provide sufficient proof that a connection exists and that data can successfully transfer from point to point.

### 3. *traceroute*

*traceroute* tracks transmission routes through Internet gateways (or it finds the miscreant gateway that's discarding your packets). *traceroute* utilizes the IP protocol "time-to-live" (TTL) field and attempts to elicit an ICMP TIME\_EXCEEDED response from each gateway along the path to some host. The *traceroute* man page is included as Appendix D.

*traceroute* is a vital part of routing problem diagnosis. The address or domain name of the end host must be known. No routing information needs to be known in advance. Until

the user becomes familiar with *traceroute* the results may be cumbersome and slow in coming. The hosts listed in the *traceroute* path are either IP numbers or domain names or any combination. Below is a *traceroute* from NPS to UCSC. The results are of value to provide a familiarity with at least some of the intermediate gateways.

```
% traceroute cse.ucsc.edu
traceroute to cse.ucsc.edu (128.114.7.12), 30 hops max,
      40 byte packets
 1  131.120.54.1 (131.120.54.1)  1 ms  1 ms  1 ms
 2  131.120.254.20 (131.120.254.20)  3 ms *  2 ms
 3  epsilon.nps.navy.mil (131.120.252.101)  4 ms  3 ms  4 ms
 4  131.120.251.51 (131.120.251.51)  7 ms  4 ms  4 ms
 5  paloalto-cr1.bbnplanet.net(131.119.2.53)  9 ms 19 ms 10 ms
 6  santacruz-cr1.bbnplanet.net(131.119.78.146) 17 ms  40 ms 62 ms
 7  U-SURE-R-NOSEY.UCSC.EDU(128.114.100.250) 45 ms 18 ms 19 ms
 8  comm-g.UCSC.EDU (128.114.1.252)  18 ms  32 ms  18 ms
 9  applsci-g.UCSC.EDU (128.114.132.253)  25 ms  17 ms  17 ms
10  arapaho.cse.ucsc.edu (128.114.7.12)  18 ms  30 ms  40 ms
```

This is an excellent example of *traceroute*. All the names and all the IP numbers of every host on the route are listed along with round trip information from each one. Degraded cases have large blocks of data missing with no explanation. It may also continue indefinitely if not set with a timeout.

#### **4. *nslookup***

*nslookup* is an interactive program to query Internet domain name servers. The user can contact servers to request information about a specific host, or print a list of hosts in the



domain. *nslookup* is not a monitoring tool per se but it warrants attention because it is an integral part of network problem debugging. The *nslookup* man page is included as Appendix E.

*nslookup* is a powerful tool. Almost any network address or alias can be obtained through the proper use of *nslookup*. Unfortunately, *nslookup* is so complex that the majority of network users are not proficient in its use. *nslookup* has two modes, interactive and non-interactive. The non-interactive version merely returns the name of a host when given the IP number as an argument and vice-versa. Additionally the server information relevant to the host is also provided.

```
% nslookup cadet
Server:  azure-63.stl.nps.navy.mil
Address: 131.120.63.1

Name:    cadet.stl.nps.navy.mil
Address: 131.120.64.17
```

The interactive version is a complex program whose behavior is partially dependent upon the configuration of the network operating system. Therefore, results vary from network to network. The following options that are available within *nslookup* emphasize its complexity.

```
> help
Commands: (identifiers are shown in uppercase,
[] means optional)
```

NAME print info about the host/domain NAME

using default server

NAME1 NAME2 - as above, but use NAME2 as server

help or ? - print info on common commands;

set OPTION - set an option

- all - print options, current server and host
- [no]debug - print debugging information
- [no]d2 - print exhaustive debug information
- [no]defname - append domain name to each query
- [no]recurse - ask for recursive answer to query
- [no]vc - always use a virtual circuit
- domain=NAME - set default domain name to NAME
- srchlist=N1[/N2/.../N6] - set domain to N1 and search list to N1, N2, etc.
- root=NAME - set root server to NAME
- retry=X - set number of retries to X
- timeout=X - set initial time-out interval to X seconds
- querytype=X - set query type, e.g., A, ANY, CNAME, HINFO, MX, NS, PTR, SOA, WKS
- type=X - synonym for querytype
- class=X - set query class to one of IN (Internet), CHAOS, HESIOD or ANY
- server NAME - set default server to NAME using current default server
- lserver NAME - set default server to NAME, using initial server
- finger [USER] - finger the optional NAME at the current dflt host
- root - set current default server to the root
- ls [opt] DOMAIN [> FILE] - list addresses in DOMAIN (optional: output to FILE)
- a - list canonical names and aliases
- h - list HINFO (CPU type and operating sys)

```
-s - list well-known services
-d - list all records
-t TYPE - list records of the given type (eg, A,CNAME,MX, etc)
view FILE - sort an 'ls' output file and view it with more
exit - exit the program, ^D also exits
```

As shown, *nslookup* performs numerous functions. It can list every node on a network that has an IP address. It can list the servers only. It can trade IP number for domain name and vice-versa. It's syntax is tricky and can be misleading. The following is an example of interactive *nslookup*. Interactive *nslookup* is entered by typing *nslookup* with no arguments.

```
% nslookup

> ls -a stl.nps.navy.mil
[azure-63.stl.nps.navy.mil]
loghost          azure.stl.nps.navy.mil
kerberos         spot.stl.nps.navy.mil
gate-azure       azure-63.stl.nps.navy.mil
periwinkle       blackand.stl.nps.navy.mil
gate-kerberos    gate-spot.stl.nps.navy.mil
www              azure.stl.nps.navy.mil
ftp              azure.stl.nps.navy.mil
```

After *nslookup* is entered, the user has all of the options listed in the *nslookup* help menu available. In the above example, the command 'ls -a' yields canonical names and

aliases of the domain servers. This is very useful information for any user to have at his fingertips.

### **5. *ttcp/nttcp***

*ttcp/nttcp* times the transmission and reception of data between two systems using the UDP or TCP protocols. It differs from common “blast” tests, which tend to measure the remote inetd as much as the network performance, and which usually do not allow measurements at the remote end of a UDP transmission. Because *ttcp* is not a packet filter, it is not a reliable measurement of the connection performance. It is a good exercise to test a connection from both ends of the ‘pipe’. The *ttcp* man page is included as Appendix F.

*ttcp* and *nttcp* are the same test but performed across different protocols. *ttcp* is used on IP networks and *nttcp* is used on ATM. The terms will be used interchangeably realizing the existence of these differences. The results are similar.

*ttcp* is useful provided its limitations are known. It requires proper installation and significant training prior to use. The transfer rate information it displays is not necessarily accurate for reasons similar to those given for *ftp*. When using *ttcp* between two networks, certain parameters must be set at both ends. This requires either an account on the network or collaboration between trained users. *ttcp* is covered fully in Chapter 4.

### **6. *tcpdump***

*tcpdump* is a packet filter. Packet filters have the ability to look into any and all packets traveling on a network. This ability greatly enhances network monitoring. Without it, true performance evaluation is not possible. Packet filters when used maliciously are a

major security violation; therefore, they belong only in the hands of experienced administrators who have the responsibility of determining accurate throughput measurements. *tcpdump* is the packet filter of choice. *ipman* and *etherman* are publicly available packet filters that provide a graphical interface. [Shulze, Farrel, 96] The man page for *tcpdump* is included as Appendix G.

*ttcp* and *ftp* both return transfer rate information but they are not packet filters. Because they are not, they are the wrong tools to measure throughput. The layered protocol model is meant to hide physical layer anomalies from the user and that is exactly what it does. Because of this, any throughput rate that is derived at a different layer is suspect. It is impossible to get a reliable measurement of throughput using *ttcp* and *ftp*.

*tcpdump* has the ability to look into every packet in a transmission. It can filter masked packets to avoid saturation and packet loss. From the packets filtered, the start time, duration, participating hosts, application protocol and bytes transferred in each direction can be determined. [Paxson, 96]

*tcpdump* provides tremendous utility. It is a vital performance monitoring tool. It must be used only by system administrators, but it must be used.

#### **D. IP MONITORING**

Internet Protocol (IP) is well understood but the layers that rest on top of IP are becoming more and more complex. There are numerous ways of putting IP packets across a physical medium. All of them must be monitored. There is essentially no difference in the ability to monitor various networks such as Ethernet, token ring and Fiber Distributed Data

Interface (FDDI) although the results will surely vary. Similarly, differences in network configuration amongst common network structures will not alter monitoring capabilities. The main thing in IP monitoring is to accurately measure and verify the results.

Network users greatly benefit from comprehensible and understandable feedback on the performance of their particular application over transmission path regardless of the type of network in use. The great majority of networks in existence use the IP over a shared-medium network such as Ethernet or FDDI. ATM networks are becoming more popular due to their advertised ability to deliver high-bandwidth, low-latency selectable quality-of-service internetworking.

#### **E. ATM MONITORING**

ATM is a switched-network mechanism for transferring cells of fixed length across virtual connections at high bandwidth and low latency. Connections can be configured on the fly through switched virtual circuits (SVCs). While the final outcome of ATM network monitoring is the same as other technologies, (i.e., network connectivity and performance), the method of determination is different. Several fundamental differences exist that impact ATM monitoring approaches including bandwidth-allocation methods, connection-oriented verses connectionless protocols, and configuration-layer variances.

Because ATM is a complex switching protocol intended to deliver high bandwidth, the network issue of most interest is transfer rate. ATM transfer rate depends on a number of factors including the bandwidth of the connecting medium and interfaces, the allocation of the bandwidth through SVC and PVC configuration and real-time congestion management.

*nttcp* can be used to measure throughput but as previously mentioned, the results are only approximate. The measure of ATM throughput is tricky. Remote Monitor (RMON) Management Information Base (MIB) is a promising technology that may answer the ATM monitoring dilemma. RMON MIB proponents expect to automate LAN diagnosis and disaster recovery and provide detailed performance analysis. The ability to accurately monitor ATM throughput is crucial to the successful advance of ATM internetworking.

## **F. MULTICAST BACKBONE MONITORING**

Multicast Backbone (MBone) monitoring is especially important because the MBone is designed for real-time many-to-many streaming applications. Abnormal behavior must be located and overcome promptly because there is little or no troubleshooting time accompanying real-time applications. Several tools exist for the purpose of MBone monitoring but certain limitations remain. MBone relies on Multicast routers (*mrouters*) and hosts which run the MBone daemon called *mrouted*. Newer routers incorporate the ability to route Multicast as well as unicast packets, however many of the routers in service do not provide native Multicast support.

An MBone audio/visual session typically needs at least 128 Kbps to provide adequate quality. Home users are not able to use MBone unless they have an ISDN connection or better. The MBone applications currently run on UNIX, Windows 95, and LINUX operating systems. MBone is in a period of rapid change. Different versions of *mrouted* can cause MBone routing snags. The first thing to check when troubleshooting MBone is the *mrouted* version. The version number is given by *mrinfo*.

## 1. *mrinfo*

*mrinfo* attempts to display the configuration information from the mrouter. The mrouter tested verifies the status of its neighbors. The IP number and name of the queried router is displayed, followed by the IP number and name of each Multicast-connected neighbor. In square brackets appears the metric cost of connection, and the threshold or Multicast time to live (ttl) is displayed. If the queried mrouter has a newer version of mrouterd, the version number, the type (tunnel, srcrt) and status (disabled, down) of the connection is displayed. If the queried mrouter does not respond, lack of feedback indicates mrouterd is not running or the host is down/unreachable. The number and a list of their neighboring Multicast addresses is part of that response. The *mrinfo* man page is included as Appendix H. The following is a test of the NPS mrouter:

```
% mrinfo 131.120.254.59
```

```
131.120.254.59 (MBone.nps.navy.mil) [version 3.8,prune,genid,mtrace]:  
131.120.53.21 -> 0.0.0.0 (local) [1/1/querier/leaf]  
131.120.254.59 -> 131.120.254.57 (star.nps.navy.mil) [1/1]  
131.120.254.59 -> 131.120.254.222 (zeta.nps.navy.mil) [1/1]  
131.120.254.59 -> 131.119.244.11 (utumno.barrnet.net) [1/32/tunnel]  
131.120.254.59 -> 192.31.48.211 (192.31.48.211) [1/32/tunnel/down/leaf]  
131.120.254.59 -> 134.89.64.1 (algae.mbari.org) [1/8/tunnel/leaf]  
131.120.254.59 -> 131.120.142.126 (pine.nps.navy.mil) [1/1/tunnel/down/leaf]  
131.120.254.59 -> 131.120.149.55 (intruder.aa.nps.navy.mil) [1/1/tunnel/leaf]  
131.120.254.59 -> 131.120.57.3 (ntc.nps.navy.mil) [1/1/tunnel/down/leaf]  
131.120.254.59 -> 131.120.141.100 131.120.141.100) [1/1/tunnel/down/leaf]  
131.120.254.59 -> 131.120.140.62 (noise.usw.nps.navy.mil)  
[1/1/tunnel/down/leaf]  
131.120.254.59 -> 131.120.151.221 (indigo1.me.nps.navy.mil) [1/1/tunnel/leaf]
```



```

131.120.254.59 -> 131.120.20.39(chandra.ece.nps.navy.mil)
    [1/1/tunnel/down/leaf]
131.120.254.59 -> 131.120.150.143 (auvonyx.me.nps.navy.mil)
    [1/1/tunnel/down/leaf]
131.120.254.59 -> 131.120.211.3 (131.120.211.3) [1/1/tunnel/leaf]
131.120.254.59 -> 198.189.249.186(MBONEIndy.monterey.edu)
    [6/16/tunnel/down/leaf]

```

*mrinfo* gateway allows us to see the configuration details of any mrouter. The NPS firewall restricts the use of *mrinfo* such that it is only available for the campus-wide mrouter.

## 2. map-mbone

*map-mbone* is essentially the same as *mrinfo* except that *map-mbone* takes *mrinfo* one step further. A neighboring mrouter responds to the initiating mrouter in the same manner except that it provides the initiating router with a list of the neighbor's neighbors. This search for unique mrouter will continue until no new neighboring mrouter are reported. The *map-mbone* man page is included as Appendix I. *map-mbone* starting from the NPS mrouter is demonstrated below.

```

% map-mbone 131.120.254.59
131.120.53.21 (mbone.cc.nps.navy.mil): alias for 131.120.254.59
131.120.254.59 (mbone.nps.navy.mil): <v3.8>
131.120.254.59: 198.189.249.186 (MBONEIndy.monterey.edu) [6/16/tunnel/down]
131.120.211.3 [1/1/tunnel]
131.120.150.143 (auvonyx.me.nps.navy.mil) [1/1/tunnel/down]
131.120.20.39 (chandra.ece.nps.navy.mil) [1/1/tunnel/down]
131.120.151.221 (indigo1.me.nps.navy.mil) [1/1/tunnel]
131.120.140.62 (noise.usw.nps.navy.mil) [1/1/tunnel/down]
131.120.141.100 [1/1/tunnel]
131.120.57.3 (ntc.nps.navy.mil) [1/1/tunnel/down]
131.120.149.55 (intruder.aa.nps.navy.mil) [1/1/tunnel]
131.120.142.126 (pine.or.nps.navy.mil) [1/1/tunnel/down]
134.89.64.1 (algae.mbari.org) [1/8/tunnel]
192.31.48.211 [1/32/tunnel/down]
131.119.244.11 (utummo.barrnet.net) [1/32/tunnel]
131.120.254.57 (star.nps.navy.mil) [1/1]
131.120.254.222 (zeta.nps.navy.mil) [1/1]
131.120.53.21: 131.120.53.21 (mbone.cc.nps.navy.mil) [1/1/querier]

```

### 3. *mtrace*

*mtrace* prints a Multicast path from a source to a receiver. A trace query is passed hop-by-hop along the reverse path from the receiver to the source, collecting hop addresses, packet counts, and routing error conditions along the path. Then the response is returned to the requester. The *mtrace* man page is included as Appendix J.

*mtrace* suffers from the inherent flaw that when you need it (i.e. when a connection is not functional), it cannot work. This is because *mtrace* starts at the address given at the command line then traces the route taken back to the originator. Therefore, when the link is not working *mtrace* will not be able to get to the location where the break is. *mtrace* requires *mrouted* version 3.3 or later. The example below demonstrates the wealth of information that is displayed by a functional *mtrace*. The results demand careful study.

```
% mtrace 134.89.64.1

Mtrace from 134.89.64.1 to 131.120.64.19 via group 224.2.0.1
Querying full reverse path... * switching to hop-by-hop:
 0 slate.stl.nps.navy.mil (131.120.64.19)
-1 zeta.nps.navy.mil (131.120.54.1) PIM thresh^ 0
-2 *
Route must have changed...
-1 cadet.stl.nps.navy.mil (131.120.64.17) DVMRP thresh^ 1
-2 ? (131.120.211.3) DVMRP thresh^ 1
-3 mbone.nps.navy.mil (131.120.254.59) DVMRP thresh^ 1
-4 * algae.mbari.org (134.89.64.1) DVMRP thresh^ 8
-5 algae.mbari.org (134.89.64.1)
Round trip time 39 ms
```

Waiting to accumulate statistics... Results after 10 seconds:

Source	Response Dest	Packet Statistics	For	Only For	Traffic
134.89.64.1	224.0.1.32	All Multicast Traffic From 134.89.64.1			
v	/	rtt 26 ms Lost/Sent = Pct Rate To 224.2.0.1 224.2.0.1			
134.89.64.1	algae.mbari.org				
v	^	ttl 8	-1/32 = -2%	3 pps	0/0 = --% 0 pps
131.120.254.59	mbone.nps.navy.mil				
v	^	ttl 9	-2/737 = 0%	73 pps	0/0 = --% 0 pps
131.120.211.3	?				
v	^	ttl 10	536/536 =100%	53 pps	0/0 = --% 0 pps
131.120.64.17	cadet.stl.nps.navy.mil				
v	\	ttl 11	1	0 pps	0 0 pps
131.120.64.19	131.120.64.19				
Receiver	Query Source				

## G. TECHNIQUES OF NETWORK MONITORING

### 1. Cron Daemon Scripts

Scripts can be executed by the UNIX command *crontab*, which is invoked with an argument of either the name a file containing the necessary *crontab* commands or standard input if no file is specified. The input to the *crontab* command are six fields, five containing timing information and the sixth containing the executable instruction. *crontab* copies this information into a log file in */usr/lib/cron*. A user's *crontab* file lists commands that are to be executed on behalf of that user at the times specified. *crontab* scripts are a flexible and powerful part of continuous network monitoring.

*crontab* is laden with subtle traps. The *crontab* command launches the default shell which may or may not yield results similar to other shells in which the executable has been tested. Alternate shells can be invoked if properly specified in the *crontab* file. *crontab* will be initiated in the users home directory unless a change of directory is specified as part of the *crontab* execution. If it is initiated elsewhere without explicit directory information, it may not function correctly, or at all. Users receive little or no feedback of any kind after *crontab* is invoked. The *crontab* man page is included as Appendix K.

## 2. Automatic Monitoring

Automatic monitoring provides continuous monitoring of designated hosts, network administration notification of changes in the network status as they occur, and the automatic documentation of system status.

Several options present themselves as a user interface to both status reporting and interactive testing. The main downfall of system generated tests and results is their poor interface. Command line interaction is cryptic at best. System files are buried in the labyrinth of administrative directories. To make monitoring worthwhile the interface must be user friendly.

HTML coupled with any web browser is the ideal user interface to automated monitoring. Home pages can be generated automatically on a specified schedule. The vast majority of users can effectively use Netscape or any other browser. HTML is totally portable and mostly standardized.

NPS network monitoring is displayed on a dynamically created HTML page with the time of the test, the name of each host tested and the status of the hosts. The product of testing is listed network by network. Lists of hosts to test reside in a separate files that can be accessed easily and changed as required without altering the monitoring script. The page is updated via a *crontab* script from a designated host. Further features can be designed into the script as individually desired.

The use of HyperText Transport Protocol (HTTP) in combination with the Common Gateway Interface is key to an effective monitoring interface. [Aoki, Brenner, 96]  
Portability, accessibility, and user capability are greatly simplified through the use of HTTP.

Proprietary solutions are not required when using HTTP. Additional hardware and software investment is not required. Real-time network status is always available through a Web browser which is an application that most users are already familiar with.

### **3. Interactive Monitoring**

Automated network status verification provides data that is relatively current. It is nonetheless historic; therefore, users must be able to check host status on an interactive, instantaneous basis. In a perfect world, all network users might know how to do this. Almost all network users do not. Operational requirements, deadlines, staffing shortages, decreased funding levels and countless other reasons force training to be stripped, sidestepped or abandoned altogether. The result is a decrease in overall user proficiency.

Network administrators are well aware of this common problem. The classic approach given this circumstance is to limit user access to monitoring tools. [Cochran, 96] Constructing artificial limits is not only deleterious to the user, it effectively forces network administrators to bear the brunt of all network debugging endeavors. Automation of network monitoring tools is a solution to this problem. It leaves the tools in the hands of the user while at the same time removing the burdens of use.

For example, the *ping* man page states that *ping* can be "extremely stressful on a network and should be used with caution." On the contrary, *ping* is a tool that all users need to be familiar with. Through an interactive interface, the options used with *ping* are totally controlled and user-proof thus limiting the stress that it can cause while still making results available.

True real-time monitoring is accomplished through an interactive monitoring interface that coexists with the automatic process. Through the interactive process, users possess the ability to verify status of any hosts simply by using any web browser. The interactive test quickly produces accurate results without command line interaction.

CGI scripts provide an ideal mechanism to create and execute interactive monitoring. The most common and most useful tests are *ping*, *traceroute* and *nslookup*. These tests lend themselves well to automation. Since network assets shift occasionally, network variables such as the resident and testable hosts need to be easily reconfigured. For the sake of simplified alteration, network assets reside in text files configured apart from the monitoring scripts.

*ping* tests the status of any host, including perhaps the local host. In practice, two live hosts at the minimum are required for a successful *ping*. If there are more nodes in the middle, they all must be alive for the *ping* to successfully complete. When a *ping* is automated, it actually completes several tests: test of the *pinged* host, test of the *pinging* host and test of the intermediate routers, (which may vary). If a server is available, it makes a good candidate for the *pinging* host. Servers function at higher demand than do other network hosts so further server tasking requires careful consideration. The use of the server as the *pinging* host is beneficial for two reasons. First, servers have to be reliable. Second, the results of the status verification may need to be stored elsewhere which means that they may need to transit the server anyway. As shown in Figure 4.1, there are no points of failure involved with the testing of host1 from server1 other than server1 and host1 themselves. If

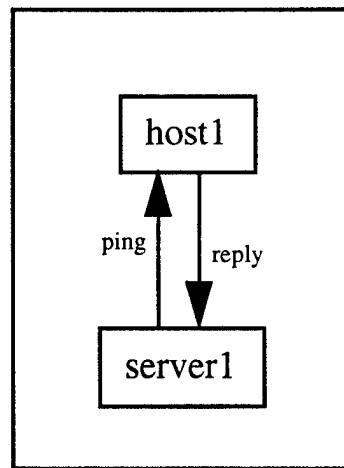


Figure 4.1. simple *ping*

an alternate host had been used to *ping* host1 rather than server1, failure could occur at either the alternate host or server1.

As shown in Figure 4.2, a *ping* from host1 to host2 which crosses network boundaries involves two hosts and two routers over six transmissions. A failure of any single transmission will result in an inconclusive test.

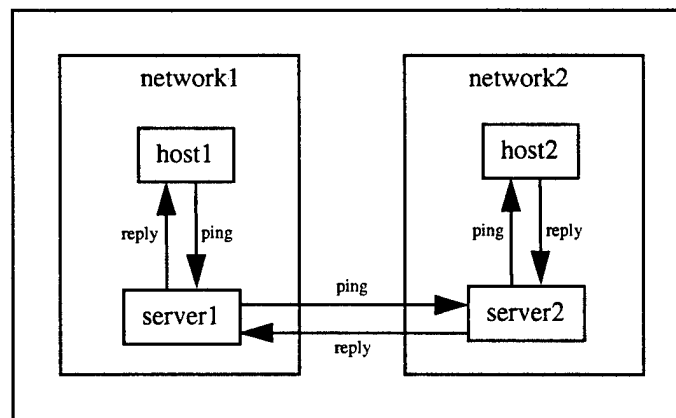


Figure 4.2. *ping* across network boundaries

The path of transmission is important to network monitoring but it is somewhat elusive. It is not under the control of network administrators but by those outside the bounds of individual networks. Routers have the ability to reroute traffic as required to enhance network performance and avoid broken links. Because of this, the exact path that a packet travels cannot be assumed.

There are numerous problems that can cause routing problems. Loop backs or continuous loops, total outages, link, router or host failure are the major causes of rerouting. The exact cause is sometimes untrackable. Two thirds of the paths through the Internet remain unchanged over periods of days to weeks. The other one third changes on time scales from seconds to hours. The probability of encountering a significant routing problem was 3.4 percent at the end of 1995. [Paxson, 96]

*traceroute* is used to obtain routing information. *traceroute* returns the hop-by-hop information of a packet from the point of initiation to the selected host along with timing statistics. *traceroute* yields the real-time path information. This is essential information when troubleshooting routing problems, unexplained performance decreases, and lack of network response. *traceroute* furnishes the information required to pinpoint these problems.

*traceroute* is a likely choice to automate because it is not easy to use from the command line. Scripting also relieves the necessity to know the exact name or address of the other end of the connection. User may not be familiar with the remote LAN on the other end of the *traceroute* at all. Automated *traceroute* can present choices in a point-and-click manner, thus alleviating the requirement to determine the specific network host information or command syntax.



*nslookup* is one of the best tools available. It is also one of the most complex. It is very useful but only on an occasional basis so users rarely have the correct syntax at their fingertips. The syntax is extremely prone to error. This complexity leads to underutilization. Thus *nslookup* is another excellent tool to automate. Automated *nslookup* presents simplified choices to the user for selection without any requirement for syntax knowledge. The only user requirement is the ability to navigate a web browser.

#### 4. Common Gateway Interface

“CGI (Common Gateway Interface) is the interface between your Web site's HTTP server and the other resources of your server's host computer. CGI is not really a language or a protocol in the strictest senses of those terms. It's actually just a set of commonly-named variables and agreed-upon conventions for passing information back and forth between the client (that's your Web browser) and the server (the computer that sends Web pages to the client). When viewed in this admittedly simplistic light, CGI becomes much less intimidating.” [Weinman, 1995]

CGI programs can be written in any language. *sh*, *csh*, *perl*, and *C* are the most common languages. All of the monitoring scripts in this work are done in *perl*. *perl* and a well-configured CGI server are an excellent combination for use in scripted monitoring. UNIX is the most common platform for running a Web server but programs must be able to run on different operating platforms. These include Suns running UNIX, PCs running Linux, NEXT, Macintosh and others. Security requirements, network loading and access requirements all affect the manner in which the CGI server is configured. The relative importance of these factors determines the availability of the server to perform CGI functions. Server configuration is important. The performance of CGI scripts, *crontab* scripts, image map operation and timing constraints all depend on the server configuration.

Familiarity with the server CGI configuration aides in writing robust monitoring implementations.

## **5. Human Factors**

At the root of network configuration is the human engineering aspect of the networking. In all configurations, a balance exists between security and user accessibility. This philosophy is driven by the environment. Academic, financial and military institutions each have different needs that foster different environments. The climate of a network is unique and depends as much on human variables as technical specifications.

## **6. Topology**

Understanding network topology is a crucial part of network monitoring yet it is often overlooked. Transmission path determination has been relegated (rightfully), to the signaling and routing mechanism at the lowest layers of the network stack. To effectively troubleshoot problems, network and internetwork topology must be fully understood. The topology cannot be completely known in advance but a reasonable expectation can exist. Evaluating monitoring results requires some network familiarity.

Logical and physical topologies are different concepts. The logical topology is the imagined path between two endpoints, ignoring each intermediate point. The physical topology is the actual route of transmission including every bridge, router and gateway. The physical topology is important in ascertaining the exact location of a communication transmission failure. Logical topologies are most useful for planning and managing functional networks.

Topologies (both logical and physical) can change frequently. They can be altered automatically by the routing and signaling mechanism. Routing and signaling protocols are meant to change the topology when a failure in a node occurs. Topology can be changed by switch owners, network administrators or service providers or anyone who has control of a host or a switch or the physical medium. ATM switching is designed to change switch configuration to shape traffic to optimum levels. So understanding the physical topology requires an ongoing effort. Control of the topology does not belong to any single person. The best that can be achieved is to determine the variables and track them as diligently as possible.

*a. NPS*

The NPS topology is comprised of several subnets connected by a campus backbone. The backbone is FDDI. The individual networks are ATM, Ethernet and FDDI. Off-campus access is controlled by a firewall. The firewall is under the jurisdiction of the campus computer center. The configuration of the firewall can determine the performance of applications or even if they can function at all. The firewall can cause total network shutdown if overloaded. Mbone and ATM do not transit the firewall in the manner of unicast IP. As security concerns become more prevalent firewall configuration must become an integral part of network monitoring.

The NPS topology is functional but suffers from a few known problems. There is no documentation. This severely impedes the monitoring process. Effective monitoring requires some system knowledge. This is hard to come by at NPS and it usually

requires a personal inspection of the host or connection. The various subnets do not fit together well. Users find it impossible to get the big picture of the NPS campus network which is a collection of several dozen LANs. All of these aspects affect the monitoring effort in various ways. Some of them affect the topology or at least the understanding of it either directly or indirectly.

The ATM topology is separate from the campus backbone. The main advantage that this offers is that there is no chance of a spillover of cells or a mismatch of protocols with the production LANs. During the developmental stage of the ATM LAN this freedom provided simplified topology and decreased the number of administrative personnel involved in network decisions. After thorough testing it is essential to be a part of the main internetwork connection to become interoperable.

#### ***b. BayNet***

The BayNet ATM topology spawned from the CalRen educational grant. It is a DS-3 connection between several Monterey Bay area partners. They are NPS, Monterey Bay Aquarium (MBA), Monterey Bay Research Institute (MBARI), California State University Monterey (CSUMB), Moss Landing Marine Laboratory (MLML), University of California Santa Cruz (UCSC), San Jose Technical Museum of Innovation (SJTMI), Pacific Bell (PacBell) and Sprint.

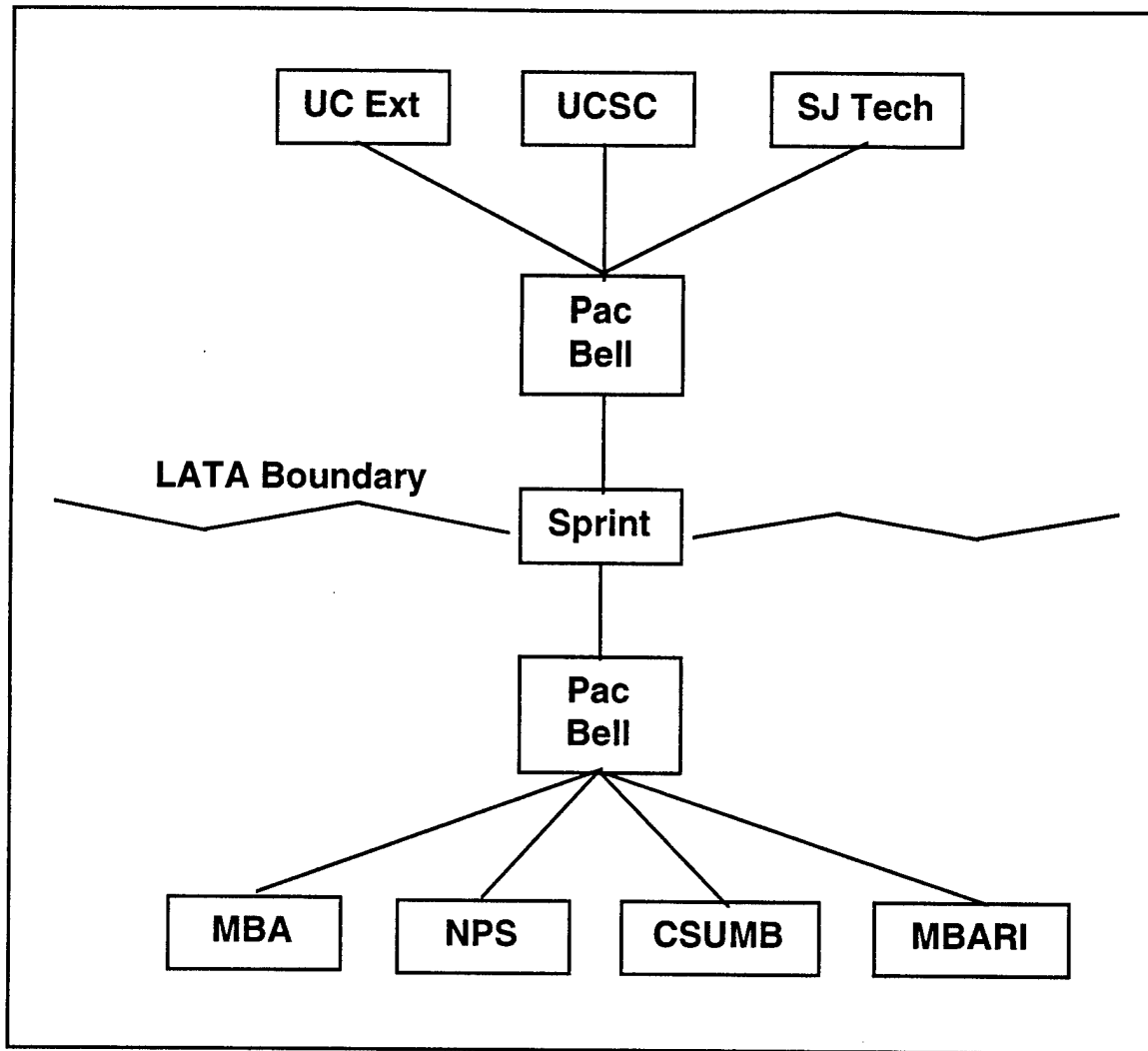


Figure 4.3. BayNet ATM Physical Topology

The physical topology of BayNet in Figure 4.3 elucidates several points. Traffic from the southern LATA to the northern LATA must pass through three switches, two PacBell and one Sprint. Intra-LATA traffic must always go through a PacBell switch. This means that any off-campus configuration requires PacBell interaction. Each iteration involving another entity introduces a delay of its own. To alter a connection between UCSC

and NPS requires interaction with at least four administrative entities. Internetwork administration quickly can become a major impediment to progress.

### *c. Beyond BayNet*

As the BayNet topology demonstrates, monitoring complexity increases with size. The number of possible contingencies increases exponentially as each node is added. Routing and signaling problems are rarely visible. When connections are lost they are quickly regained, usually transparently. When the scope of the network increases to national level, the difficulty in tracking problems seems insurmountable.

The most important part of national and global internetwork monitoring is that each network have their own domain in order. Limiting the unknowns on each end of the connection is how to control and monitor the strength of the connection. Internetwork monitoring is a test of the connection not a test of the reliability of the terminating hosts. While it is impossible to control the variables that contribute to the routing mechanism of national transmissions, having stable networks on opposite ends greatly enhances the possibility of quickly and accurately determining when and where a failure occurs.

## **7. Time**

The significance of operating system computation of the current time cannot be overstated. Much of the previously discussed material depends on accurate and stable time. Minor variance in the current time between various hosts and networks is inevitable. Hopefully the difference will be small. Operating system corrections to the internal clock can skew data in a manner that is often untrackable. Throughput measurements made in the

presence of clock corrections or gross inaccuracies will be totally unreliable. Each operating system has a different resolution, meaning the period in which the timestamp is incremented. Global solutions to these problems can be addressed through installation of the Network Time Protocol (NTP) (Request for Comments (RFC) 1305, March 1992).

## H. CONCLUSION

Network monitoring and performance evaluation is absolutely essential to network management. Currently network monitoring capabilities at most sites are desperately lacking. Proper monitoring can be accomplished using the software tools that are currently available. Public-domain software packages, easy-to-use portable HTTP/HTML interfaces and robust CGI/perl implementations enable programmers to quickly build working solutions without the necessity of purchasing off-the-shelf monitoring systems. Packet filtering software is a must for genuine throughput measurement by network administrators.

Network monitoring is difficult. It is difficult because it requires continued diligence in measuring and analyzing network resources. It is not difficult in a theoretical sense. The components of an effective monitoring effort are active and passive measurement, as well as automated and interactive testing. After LAN behavior becomes familiar and well documented the monitoring effort can easily move outward toward global Internet connections.

Every network user does not have to understand the full workings of the Internet. Most users can understand their own domain out to it's border. As network entities adopt the

mentality of keeping there own piece of the monitoring equation in order, the problems of Internet growth and control diminish.



## **V. NETWORK MONITORING TECHNICAL DETAILS**

### **A. INTRODUCTION**

This chapter emphasizes the many considerations on the frontier of network monitoring. Due to the inherent requirement for military operations to be self-sufficient the DoD has a particular interest in the broad integration of network monitoring. Network monitoring is not just important to large entities like the DoD but to every organization attached to the Internet. Public-domain applications can fulfill a large portion of the monitoring solution when used wisely. The technical elements of these tools are covered in detail in this chapter. Address and pathology resolution are vital to any monitoring endeavor and will be fully covered as well. The automated and interactive interfaces to NPS monitoring are HTML pages. These represent a user-friendly way for all network operators to monitor their LAN. Finally, because of the commonality of HTML, the monitoring capability developed in this thesis is fully portable as demonstrated throughout the BayNet region.

### **B. MONITORING CONSIDERATIONS**

Network monitoring and management are significant. Users do not often concern themselves with network performance until it slows down, then every user becomes keenly interested. It is in such critical periods that monitoring becomes visible. It is no less important however through all phases of networking and internetworking.

The success that networks enjoy today is a consequence of intense efforts in monitoring and management during their design and implementation. Monitoring of network status and performance is always important but it is much more evident during the introduction of a network technology; often disappearing altogether when working systems deploy. At the forefront of network technology today is ATM. ATM proponents promise high-bandwidth performance and guaranteed quality of service while detractions such as complexity and cost continue to surface. At the center of the entire controversy is the question, "what can ATM really do?" Millions of dollars hang in the balance, decisions that will affect millions of people must be made, and the results will be with us for decades to come. Nevertheless, performance is poorly monitored and rarely verified. Despite such curious disconnects between cause and effect, it is rarely recognized that monitoring is important.

Monitoring can mean many things to different people. Measurements can occur at different network protocol layers. Any measurement not taken at the physical layer is suspect. For example, throughput may be reduced to nothing more than a measurement of the number of packets leaving the TCP layer of the local host. The packet loss at this level may be zero because the network is not included in the measurement when in reality not one packet made it to the physical medium. Manufacturers will often tout zero packet loss and high throughput figures when representing their equipment. To accurately compute meaningful statistics (such as those discussed in Chapter III), a packet filter is absolutely essential.

The answer to the complex performance monitoring problem is that there is no single simple answer. The only accurate results will be obtained through precise measurement and analysis. It is a tough but worthwhile endeavor. The following quote puts the problem in perspective.

“Not having these important answers puts network managers in the position of a car buyer who assesses a potential vehicle purchase according to the speeds shown on the speedometer of the shiny new auto. Although the manufacturer may have created a speedometer that shows speeds up to 120 mph, doing so is no guarantee that the car can actually travel that fast, or can travel safely at that speed.” [Krivda, 95]

Accurate monitoring is not easily obtained. Technological advances in transfer rates are not matched by a corresponding increase in ability to measure. It remains difficult to consistently, accurately and effectively measure network performance. Network monitoring has been left behind in the frenzy of technological advance. But the basics of network management are essential nonetheless. It is analogous to a good navigator who never stops dead reckoning regardless of the myriad electronic navigation aids available. The good networker will always possess the ability to measure and verify the status of the network and the performance of the connection.

### **C. MONITORING IN THE DEPARTMENT OF DEFENSE**

The emphasis of this thesis is on readily available public-domain software. Public-domain tools get little attention in the technologically charged and budget-oriented climate of the DoD. Procurement efforts focus on state-of-the-art systems. Organizations with billion-dollar budgets find it difficult to adopt and rely on freeware or shareware. In the field of network monitoring as most other areas, advanced commercial

hardware and software solutions are available. But are these 'advanced' solutions necessary? Public-domain tools are proven winners when they exist for several years and field wide acceptance. The numerous drawbacks of commercial monitoring mechanisms may prove to be the catalyst required to overcome the urge to 'buy' a solution.

The most obvious drawback of commercial systems is cost. Off-the-shelf cost however is not the only concern, or even the most significant one. System monitoring integration requires training. The continued use of a proprietary tool requires maintenance, management, upgrade compliance and troubleshooting. Thus life-cycle management must be considered. Furthermore, the commitment to this equipment places the responsibility for network monitoring solely on the network administrator. Awareness of network behavior thereby eludes the average user. It is like asking someone to drive a car but never look at the dashboard at all.

DoD is rightly moving away from 'stovepipe' or stand-alone systems because of the associated life cycle considerations. That trend coupled with the genuine need for network monitoring among network users and administrators alike leaves a void. One way to fill that void is to prudently implement basic networking tools wrapped in intelligently designed software in a robust and well configured environment. This thesis provides proof-of-concept versions of such tools.

#### **D. PUBLIC-DOMAIN SOFTWARE TOOLS**

Every network is unique. In a monitoring sense however, network philosophy is common: verify host status, determine the shape of the traffic, determine the transfer

rate, cell loss rate and location, and finally know what routers are in use, their status and the common routing pathology. The majority of the work in this thesis has been performed using the Internet Protocol (IP). IP is the most common protocol. Nevertheless the solutions presented here are relevant to other network protocols as well. The simplicity of this design, which revolves around basic tools and sound networking doctrine, minimizes the impact that protocol differences, network topology and other factors might have. The prerequisites to understanding network performance are the same on ring or star topology, connection-oriented or connectionless protocols, and switched or permanent circuits. Public-domain tools coupled with effective monitoring software are a feasible way to accomplish network monitoring without a resource dependent solution.

The technically significant points of the public-domain tools are covered below. MBone monitoring tools are covered in part. For full monitoring evaluation of MBone tools see [Erdogan, 96].

### **1. *ping***

*ping* is an essential tool in network monitoring. It is a simple test of the physical link between two points. When *ping* is used to test a connection comprised of more than one leg, i.e. there are potential points of failure in between, there is no feedback of failure location. *ping* is therefore usually only the first step in a troubleshooting effort.

*ping* behaves in different ways across platforms but from a user's perspective, these differences are more like small idiosyncrasies rather than serious dissimilarities.

How *ping* can be of benefit depends more on the network configuration than on any difference in the test itself. For instance, *ping* more accurately narrows the search for a broken link in a star topology than in a ring. *ping* performs the same test but the topology is different. ATM networks present other incongruities. These result from the addressing changes that are involved. For example, a dual-homed network with FDDI and ATM operating on the same workstations and possibly across the same fiber, require extra care to preclude an addressing mismatch. *ping* from an ATM address to an IP address will not work but there is no feedback indicating such.

## 2. *ftp*

*ftp* is useful as an integrity test of network performance. *ping* provides the basic connection status information but does not provide proof that data arrived unharmed. *ftp* is a good first test used to verify network fitness. The result of the *ftp* can indicate the integrity of the connection as noted in the *ftp* below.

```
ftp> get .cshrc
local: .cshrc remote: .cshrc
200 PORT command successful.
150 Opening BINARY mode data connection for .cshrc (5649 bytes).
226 Transfer complete.
5649 bytes received in 0.04 seconds (135.62 Kbytes/s)
ftp>
```

When troubleshooting a new or faulty connection, *ftp* is enough to verify solid connectivity. To simplify the test, use a small transfer file first. This keeps the tests expeditious and easily recognizable. Later, large file transfers can be used to stress the connection. Damage to the data (if there is damage) will then become obvious. Below, a large file *ftp* demonstrates solid connectivity and some idea of the transfer rate.

```
ftp> get n16e201.exe
local: n16e201.exe remote: n16e201.exe
200 PORT command successful.
150 Opening BINARY mode data connection for n16e201.exe (2507571
bytes).
226 Transfer complete.
2507571 bytes received in 18.30 seconds (133.82 Kbytes/s)
```

### 3. *traceroute*

*traceroute* is an essential tool in route verification and management of a network. Its display is cumbersome which makes it a likely candidate for scripted interface. The first example below is a test of the connection between NPS and UCSC from royal.stl.nps.navy.mil which is a dual-homed host.

```
% traceroute cse.ucsc.edu
traceroute to cse.ucsc.edu(128.114.7.12), 30hops max, 40byte
packets
1  azure-63.stl.nps.navy.mil (131.120.63.1) 1 ms * 1 ms
2  131.120.54.1 (131.120.54.1) 2 ms 1 ms 1 ms
```

```

3  131.120.254.20 (131.120.254.20)  2 ms * 2 ms
4  epsilon.nps.navy.mil (131.120.252.101)  3 ms  3 ms  4 ms
5  131.120.251.51 (131.120.251.51)  4 ms  4 ms  3 ms
6  paloalto-cr1.bbnplanet.net (131.119.2.53)  9 ms  1ms  9 ms
7  santacruz-cr1.bbnplanet.net (131.119.78.146) 14 ms 14 ms 15 ms
8  U-SURE-R-NOSEY.UCSC.EDU (128.114.100.250) 22 ms 18 ms 17 ms
9  comm-g.UCSC.EDU (128.114.1.252)  19 ms 16 ms 16 ms
10 applsci-g.UCSC.EDU (128.114.132.253) 18 ms 16 ms 16 ms
11 arapaho.cse.ucsc.edu (128.114.7.12)  16 ms 23 ms 18 ms

```

Without further explanation, the user can tell that the transmission was complete.

As the level of familiarity with the internetworking environment increases the more meaningful *traceroute* is. Truly bizarre routes will stand out.

This example is a *traceroute* between NPS and UCSC from royal. The host at UCSC is an ATM host.

```

% traceroute cyclone.cse.ucsc.edu
traceroute to cyclone.cse.ucsc.edu (172.20.70.2), 30 hops  max,
40 byte packets
1  * * *
2  * * *
3  * * *
4  * * *
5  * * *
CTL^C

```



Notice that *traceroute* immediately indicates the IP address which it is testing. In this case, 172.20.70.2 is the IP address of an ATM adapter on cyclone. The only way to know that *traceroute* is attempting a connection to an ATM host is if the user is familiar enough with the network to recognize that the address is in the ATM domain. It is unknown to the user whether *traceroute* is from the IP or the ATM side of the host. This test eventually had to be terminated manually as shown.

#### **4. *ttcp/nttcp***

In the introduction to this chapter, numerous inconsistencies in monitoring results were highlighted. Measurement of only the outgoing cells is a flawed test from the outset. The ability to see both ends of the pipe as in the case of *ttcp* is a much better test of the connection even though the throughput measurement is not conclusive. *ttcp* times the combined transmission and reception of data between two systems using either the UDP or TCP protocols and it allows measurements both ends of a UDP transmission.

A significant stumbling block in the use of *ttcp* is that the user must have access to both ends of the "pipe." If *ttcp* is conducted between two hosts on the same LAN then access is not usually a problem. Internetwork *ttcp* users must have access to not only their network but also to the remote network. This limits the flexibility of *ttcp* considerably. Users who have access to both ends of the link however, will be able to exercise the connection. In the following example, colum.cse.ucsc.edu is the remote host and royal.stl.nps.navy.mil is the local host.

##### **a. *set remote host to receive***

```

column 1> nttcp -r

ttcp-r: buflen=65536, nbuf=2048, port=5001 tcp
ttcp-r: socket

```

The test returns default parameters for buffer length, nbuf and port. *ttcp* is ready to receive packets conforming to those parameters. Packets that do not match this setup will cause the test to fail. There is no feedback of failure.

***b. set local host to transmit***

```

% nttcp -t colum.cse.ucsc.edu

ttcp-t: buflen=65536,nbuf=2048,port=5001 tcp ->
colum.cse.ucsc.edu
ttcp-t: socket

```

The same parameters are returned. These match the parameters set on the receive end. A novice user will expect the test to continue at this point; however, this is an example of a failed test. The cause of failure is not given. In reality, it is a failure of the ATM host at NPS. With a good connection, a valid test will return the parameters that are set at the beginning of the transmission which are *ttcp-t*: connect, send window size, and receive window size. The only way to be aware of *ttcp* failure is to be thoroughly familiar with a valid *ttcp*. The following is an example of a valid test. The local and remote hosts are both royal.stl.nps.navy.mil. This is termed a loopback test. The command *nttcp* was used vice *ttcp* as the test was conducted across an ATM network.

***c. set remote host to receive***

```
% nttcp -r
ttcp-t: buflen=65536, nbuf=2048, port=5001 tcp
ttcp-t: socket
```

***d. set local host to transmit***

```
% nttcp -t atm-royal.stl.nps.navy.mil
ttcp-t: buflen=65536, nbuf=2048, port=5001 tcp->
atm-royal
ttcp-t: socket
ttcp-t: connect
send window size = 57344
receive window size = 57344
```

***e. test completion - local***

```
% nttcp -r
ttcp-r: buflen=65536, nbuf=2048, port=5001 tcp
ttcp-r: socket
ttcp-r: accept from 172.20.70.1
send window size = 57344
receive window size = 57344
ttcp-r: 134217728 bytes in 9.30 real seconds =
14088.94 KB/sec = 112.7116 Mb/s
ttcp-r: 2353 I/O calls, msec/call = 4.05,
calls/sec = 252.92
ttcp-r: 0.0 user 3.1 sys 0:09 real 33%
```

***f. test completion - remote***

```
% nttcp -t atm-royal.stl.nps.navy.mil
ttcp-t: buflen=65536, nbuf=2048, port=5001 tcp ->
atm-royal
ttcp-t: socket
ttcp-t: connect
send window size = 57344
receive window size = 57344
ttcp-t: 134217728 bytes in 9.29 real seconds =
14103.52 KB/sec = 112.8281 Mb/s
ttcp-t: 2048 I/O calls, msec/call = 4.65,
calls/sec = 220.37
ttcp-t: 0.0user 4.9sys 0:09 real 53%
```

The results are presented after all packets have been sent. The transfer rate given is 134217728 bytes in 9.29 real seconds. The receive and transmit results are 112.7116 Mbps and 112.8281 Mbps respectively. The major concern at this point is to determine the genuine throughput and to locate whatever causes are present which decrease the throughput from the specification of 155 Mb/s.

The other numbers listed in the results confirm the buffer size, the window size and the port in use. The defaults are the best choice when running *ttcp* in most cases. The parameters must be compared with each other to ensure that they match. An important network parameter that the user must know to successfully implement this test is the Maximum Transmission Unit (MTU). For example, the Ethernet setting is 1508 bytes. If the network setting is different for some reason, the media packet size must be

set less than or equal to the network MTU. If this is not done in those networks where it needs to be, IP fragmentation (reducing packet size to less than MTU) will distort the test.

*ttcp* provides numerous switches which afford the user as much diversity as necessary to perform a thorough test. Tests lasting at least ten seconds should be used to obtain accurate measurements. There is little feedback during the test as shown but the results (when forthcoming) are valuable.

*ttcp* can also be used as a “network pipe” for moving directory hierarchies between systems when routing problems exist or when the use of other mechanisms is undesirable. A different version of *ttcp* is in place to facilitate file transfer and is quite often the default standard. The user will know when this is the case when the operating system response to a *ttcp* command is as follows:

```
Usage: ttcp [-pL] file1 file2
        ttcp [-prRL] path1 [path2 ...] dir
        ttcp -v
        ttcp -h
        -L      do not perform a cp(1)
        -v      print the version number and quit
        -h[elp] print this message
```

## **E. ADDRESS RESOLUTION**

Address resolution is vital to monitoring. Monitoring that is performed without a grasp of the manner in which hosts talk to one another is merely guesswork. Of particular importance is the interaction between IP and ATM since they use different addressing schemes.

## 1. IP Addressing

IP addressing is the best understood and most common network and internetwork addressing protocol. A thorough understanding of IP addressing is essential to understanding network monitoring and management. Subtleties exist which may cause internetwork communication problems.

For example, the System Technology Lab (STL) LAN at NPS (like many LANs), has more than one set of Class C addresses. The domain contains FDDI subnet 131.120.64.xxx, Ethernet subnet 131.120.63.xxx, ATM subnet 131.120.75.xxx, and one machine is 172.20.70.xxx which is part of the U. C. Santa Cruz subnet. Of particular interest are the machines that are dual homed. The dual-homed machines can be determined by using UNIX command `ypcat` and then piping the results to the `grep` command to match the subnet.

```
% ypcat hosts | grep .64
```

```
131.120.64.23  navy.stl.nps.navy.mil  navy
131.120.64.5   spot.stl.nps.navy.mil  spot
131.120.64.27  teal.stl.nps.navy.mil  teal
131.120.64.4   azure.stl.nps.navy.mil  azure www ftp
131.120.64.17  cadet.stl.nps.navy.mil  cadet
```

```
% ypcat hosts | grep gate
```

```
131.120.63.5   gate-spot.stl.nps.navy.mil  gate-spot
131.120.63.23  gate-navy.stl.nps.navy.mil  gate-navy
```

131.120.63.1	azure-63.stl.nps.navy.mil	gate-azure
131.120.63.27	gate-teal.stl.nps.navy.mil	gate-teal
131.120.63.17	gate-cadet.stl.nps.navy.mil	gate-cadet

This presents the error-prone situation that one machine may possess two or more host names, (such as 'navy' and 'gate-navy') and have addresses that are very similar.

The STL ATM switches can be configured with 12 different host addresses each.

Because of the sophistication of network operating systems, there is no visible seam between FDDI and Ethernet, at least from a user's standpoint. It is quite easy in this situation to overlook an address mismatch.

## **2. ATM Addressing**

IP is expected to be the dominant protocol of internetworking for the near future.

The current version, IPV4 will be superseded due to it's 32 bit address format. IPV6 is the approved standards-trade replacement. [Deering, 96] ATM is also a contender for combined local and wide-area networking. Despite reports of it's demise, ATM is active worldwide. It therefore makes sense at this point to be prepared for the integration of ATM. Regardless of the Internet protocol politics, ATM is unquestionably an available technology with the proven capability to deliver high bandwidth, low latency network traffic. [53 BYTES, 96] Unfortunately ATM also has significant flaws. [Courtney, 96]

ATM addressing is complex. The first step in determining the behavior of an ATM network is to understand the basics of ATM. Then it is essential to understand how it meshes with other protocols.

ATM is a connection-oriented network layer technology. ATM offers the possibility of global connectivity using high-speed switching technology and fiber optic media, thus delivering high bandwidth and low error rates. ATM also makes promises of on-demand bandwidth allocation, quality of service (QoS) guarantees, and traffic support for a wide range of requirements. One significant obstacle to the progress of ATM is the existence of large number of IP legacy LANs. Previous network expansion did not suffer from this impediment as there was little established infrastructure. So to introduce a new networking protocol requires the justification of a significant need coupled with attendant advantages.

The decentralization of many large organizations is increasing ATM's allure. Their operations are no longer tied to traditional venues or geographical constraints. Network computing is becoming more distributed not only as a nicety but as a necessity. The standard geo-centric concept of the legacy LAN is eroding. [53 BYTES, 96]

Current IP routing requires geographically distributed nodes be assigned different IP subnet addresses. ATM however offers the ability to have distributed nodes on the same IP subnet. This is demonstrated in one of the hosts of the System Technology Lab network. Not only is host atm-royal.stl.nps.navy.mil (172.20.70.1) part of two logically distinct networks, it is able to communicate using two completely separate protocols. As shown below, royal is part of the System Technology Lab 63 FDDI LAN and the U. C. Santa Cruz ATM LAN (172.20.70.1) .



```
% ypcat hosts|grep royal
```

```
172.20.70.1      atm-royal.stl.nps.navy.mil  atm-royal
```

```
131.120.63.16   royal.stl.nps.navy.mil      royal
```

Rather than allowing only physical IP subnets, ATM allows logical IP subnets (LISs). Nodes separated by the network cloud can be part of the same LIS regardless of the number of hops needed to get from the source to the destination. Although the nodes can have the same IP subnet (Class C) address, ATM technology requires each node have a unique ATM address. [Alles, 1995]

An ATM address uses the same hierarchical concept defined by the OSI reference model Network Service Access Point (NSAP) address. As with all NSAP format addresses, an ATM address consists of two distinct parts: the Initial Domain Part (IDP) and the Domain Specific Part (DSP). The IDP for an ATM address is itself composed of two parts: the Authority and Format Identifier (AFI) and the Initial Domain Identifier. The length of the IDP and the DSP will vary depending on the NSAP address format used in implementation. However, the length of an NSAP format address cannot exceed 19 bytes. ATM addresses add a one byte selector field for local use. Each ATM network interface card is assigned a unique media access address by the IEEE just as with standard ethernet NICs. [Alles, 95]

Both the IP address and the ATM address use the MAC address as a part of the unique host address. The End System Identifier (ESI) in the ATM address is the MAC address assigned to the ATM NIC inside the computer and is used to uniquely identify a

particular node. A host on an ATM network can be a member of up to four networks simultaneously using the same NIC. [Alles, 95]

One governing uncertainty in this area is the lack of established standards thus subtle differences in addressing exist from vendor to vendor. Lack of connectivity results from a lack of standards. The System Technology Lab uses Cisco routers with Fore adapter cards. Fore, Cisco, Newbridge and NEC are all present in the BayNet area. Each of these companies employ their proprietary implementation of ATM addressing and switching. Despite the goals of the ATM Forum, interoperability cannot be assured. The critical nature of this situation cannot be understated and will be further covered in the Experimental Results and Lessons Learned Chapters.

All ATM addresses must be 20 bytes (40 characters) long. The 20 bytes can be used in one of three formats: DCC ATM Format, ICD ATM Format, or NSAP Format E.164. The NSAP address is the point "at which the OSI Network Service is made available to a Network Service user by the Network Service provider." [McKenzie 1985]. This is referring to client and server processes whether within the same end system or between two or more end systems. The NSAP address is the information the service provider (i.e., server process) needs in order to identify a particular Network Service Access Point. This information includes the value of various parameters (e.g., "called address," "calling address," "responding address," "source address" and "destination address") depending on the primitive or process (procedure) being used. The content of the NSAP address will change depending on the primitive being used. [Fore, 95]

NSAP addresses are hierarchical and are based on the concept of domains (e.g. .edu, .com, .mil, .org, .gov, etc.). Each domain can be broken down further into subdomains and sub-subdomains (e.g. navy.mil and nps.navy.mil) as required. Each domain controls a portion of the global set of all possible NSAP addresses, and each domain and subdomain (or sub-subdomain as the case may be) must have a designated "authority" to ensure the unique assignment of addresses within that domain. (The authority for the U.S. is ANSI; the authority for NPS is the Director of Computing Services.) Each authority is free to assign their set of addresses as they see fit as long as the addresses are uniquely allocated within the domain. [Dennis, 96]

An ATM address is not a network service access point (NSAP). The ATM address is in the same format as an NSAP address, but the ATM address is merely a unique identifier for a particular host on an ATM network - a network connection point, not a network service access point.

### **3. Address Resolution Protocol (ARP) Server**

ATM is a very complex technology that combines the functionality associated with the bottom three layers of the OSI reference model. ATM was originally designed for use with optical fiber at the physical layer while other complex protocols provide the functionality of the datalink and network layers. Optical fiber's reliable high bandwidth and low error rates lead to the end of substantial error and flow control problems. The emphasis was instead placed on rapid switching. ATM cells are switched/routed using hardware registers rather than memory buffering. [Alles, 95]

ATM answers the demand for the high-bandwidth, low-latency networking that upcoming 3-D graphics and video applications will require. The demand for such applications is steadily growing. Additionally it is likely that users may desire to route other protocols across an ATM network.. These networks are the LISs referred to earlier. These subnets are logical in the sense that with ATM, nodes on an IP subnet no longer need to share the same physical media. LISs don't have to be in the same geographic area but can now be separated by the network cloud. The term "Classical IP over ATM" is used to concisely describe this feature. IP traffic must be able to get from one host to another host that's a member of the same LIS but across the ATM network. [Alles, 95] Unfortunately this term is a misnomer because it only is used to refer to unicast (point-to-point) IP, ignoring Multicast (many-to-many) IP.

ATM and IP need a method of communicating and interoperating. In order to successfully send an IP message across an ATM network, the sending host must know the ATM address of the receiving host, not the IP address. An IP host must resolve IP addresses with MAC addresses, in the same way an ATM host must resolve the ATM address with the IP address. This is the purpose of ATM ARP servers and inverse ARP (INARP). The ATM ARP server is used to resolve the node's IP address with its ARP address.

The ARP server behaves like any other host except in the management of the ARP cache. The ARP cache holds address mapping. Ideally a server makes the best ARP manager. Allocating a server to manage the ARP cache is the only way to maintain

scalability. If every host had to maintain a current ARP table then each time a node was added to the network, every host in the network would have to be updated. [Fore, 95]

Network traffic does not pass through the ARP server. Address resolution is handled through a special signaling protocol - special data primitives that travel back and forth between a host and the ARP server. The resolution process is simply a table look-up. If an address is not found in the server's ARP table an ARP signal is sent out on the "net" requesting IP to ATM address resolution. In the future dynamic configuration will be the norm.

ATM cells travel along PVC's or SVC's. IP to ATM address resolution is required only in the case of SVCs. PVCs are simply established pipes that are not reconfigured on the fly. They are established well in advance through a collaborative effort of network members. PVCs must have an allocated bandwidth. Applications that require a predictable bandwidth on a regular basis running across a PVC will not be stepped on by other transmissions. The primary advantage of PVCs is that the bandwidth allocation is fixed. It cannot be shared with SVCs. This is also the primary disadvantage. It severely hampers the flexibility of PVCs thus limiting their use to special use applications. When a cell is going to travel on a PVC, the PVC number is loaded into each cell's header. There is no switching or address resolution that takes place. [Alles, 95] The NPS ARP server is not configured which leaves NPS in the untenable position of running only PVCs across the ATM network.

When using SVCs, the ARP server receives the destination information from the initiating host, resolves the address using the ARP cache, then returns the SVC

information to the host for it to load into each cell. This is achieved transparently. The entire process of setting up the network ARP server is automated via auto address configuration using Interim Local Management Interface (ILMI). Unfortunately, this process is not standardized. [Alles, 95] [Dennis, 96]

Serious pitfalls exist that may prohibit ATM from advancing to wide acceptance. ATM and IP do not operate well together and ATM compatibility with true IP Multicast is problematic. Lack of ATM standards cripples interoperability. Inflexibility decreases the ability to rely on ATM for long-haul networking ventures. Lastly, the complexity of ATM and shortage of trained experts causes uncertainty in ATM endeavors.

## **F. NPS MONITORING**

The philosophy of NPS network monitoring is both local and global. The key to successful global monitoring is to become thoroughly familiar with local network behavior. Portability and commonality are important aspects that have been preserved in design decisions. NPS monitoring is a stepping stone to effective monitoring of global internetworks.

### **1. Topology**

Network topology is one of the most important elements involved in network monitoring and management. There is no way to thoroughly comprehend network behavior without a good understanding of the expected path of transmission. There are two types of topologies: physical and logical.

The logical topology represents the virtual circuit that a packet (or cell) travels from endpoint to endpoint. The physical topology represents the exact route that each cell travels from endpoint to endpoint including switches, routers and gateways. The logical topology is usually simpler than the physical topology as cells can transit switches and routers without changing virtual paths. While the differences between the two may be subtle, they may be of great consequence. Figures 5.1 and 5.2 portray the logical and physical topologies of a simple network connection between NPS and UCSC. The topology is that of an ATM connection but similar differences between logical and physical topologies exist in other protocols.

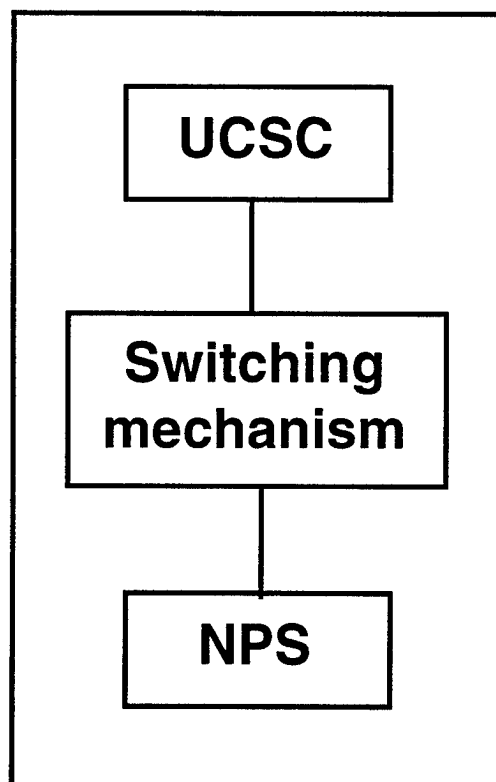


Figure 5.1. Logical Topology  
between NPS and UCSC

In an ATM connection between NPS and U. C. Santa Cruz, the cells leave NPS, are switched, then arrive at the entry point of U. C. Santa Cruz's ATM network. This is the logical topology of the connection. It's the logical way of thinking about this network connection. Generally speaking, a cell leaves a host, goes through some switching in the middle, then arrives at the destination. The exact composition of the network between endpoints is usually not an ongoing concern. The topology is likely to switch on the fly via the switching mechanism. In IP, even packets of the same transmission may take different routes.

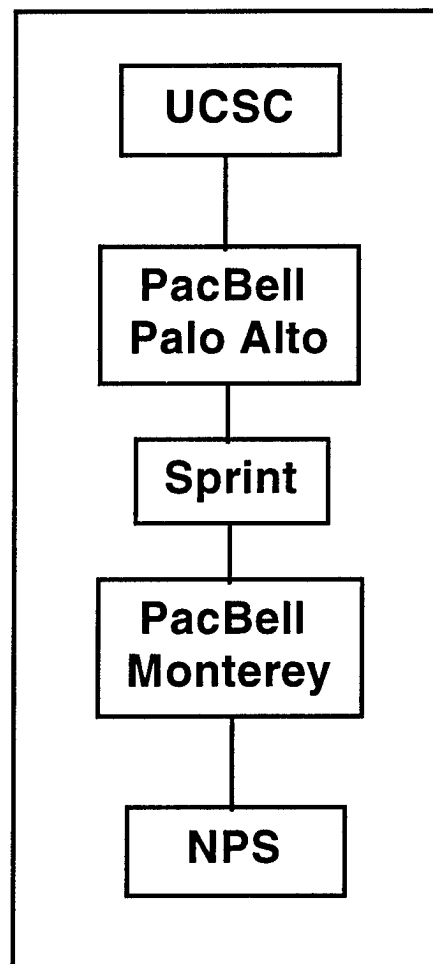


Figure 5.2. Physical ATM Topology between NPS and UCSC



The physical topology of the ATM connection between NPS and U. C. Santa Cruz shows each switch on the route. In addition to the physical topology between campuses, there exists separate physical topologies within each campus.

Troubleshooting without a solid understanding of the physical topology is a shot in the dark. While the logical topology is the way we visualize a network connection, isolating a point of failure requires knowledge of the physical topology. Physical topology is not constant so the prudent network manager will have the ability to verify it at short notice.

## **2. Automated Monitoring**

The purpose of automated monitoring is threefold. First is to verify network status on a regular basis and display those results in an easily accessible, readily understood manner. To this end, testing requires no user interaction other than that of selecting a URL via a web browser. The results are given in nontechnical form with brief explanations where required for clarity. The second purpose is to notify appropriate personnel such as network administrators or managers of changes in the monitored hosts. This must be accomplished in as unobtrusive of a manner as possible. E-mail upon initial fault discovery and when active status is regained has been chosen as the reporting trigger. Included with the e-mail that a host has recovered is a catalog of the inoperative period of that host. The final intent of automated monitoring is to provide an archive of network performance that is available to any user. The design is based upon the

supposition that users will be interested in host performance during certain time periods; therefore, the host performance is archived using filenames containing host name and down period timestamp. For standardization, the down record is very similar to the e-mail notification that a host has recovered from failure.

Continuous monitoring can be performed two ways; as a background program that executes continuously, or as an executable that is called by a *crontab* script. Monitoring calls require very little system resources and as such are well suited to the latter. The overhead of running a full time program can be avoided by monitoring using timed scripts rather than continuously executing programs.

The *ping* man page advises against using *ping* from automated scripts. *ping* used from an automated script can cause network problems when used incorrectly. When properly used however, the opposite is true. Scripts that are well written shield the user from command line syntax and protect the network user error. *ping* is a fundamental network management tool. It must be available to all users.

#### *a. Directory structure*

The monitoring software, related text files and subsequent reports need to be protected but accessible. The automatically executing monitoring code and the CGI scripts must be secure. The archive files and the host text files must be available for all users to query. Each file and the directories in which they reside must be assigned the permissions that do this. The optimum directory structure starts with a *monitoring* directory which will house all monitoring executables, related files and sub-directories.

The monitoring directory can be owned by a single user or a group. Once the monitoring tools are in place, there is little reason to access or modify the code itself. The executables therefore can be invisible to network users. The BayNet monitoring code is available via symbolic link only. Conversely, the archives and the host files reside in self-evident locations with public read permission set. Scripts that access (read/write) other files need to be able to do so. This means that either the files are visible from the same directory or symbolic links are used. 'Hard-wiring' the directory structure into the code has been avoided to enhance portability and modification. This configuration reflects the commitment to putting monitoring tools into the hands of all users while preserving the security of the system.

Figure 5.3 represents the monitoring directory structure in general. The monitoring structure can exist in it's own account or within another account. Figure 5.4 is the NPS rendition of the monitoring directory setup. As intended, the naming convention is self-explanatory

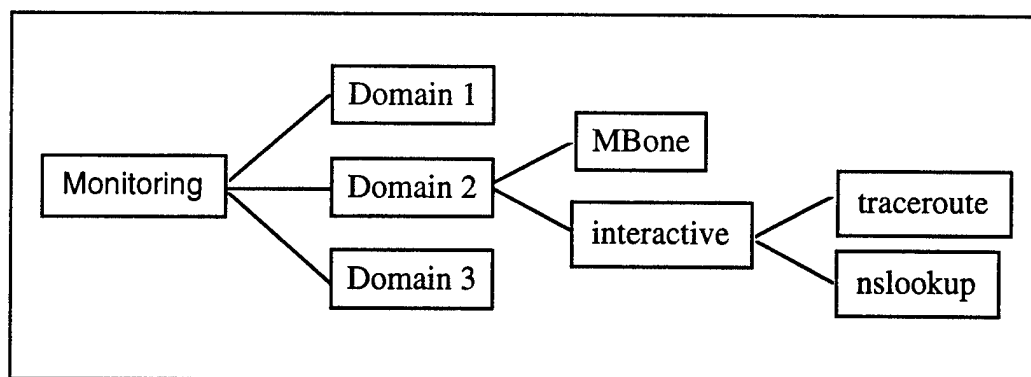


Figure 5.3. Network Independent Directory Structure

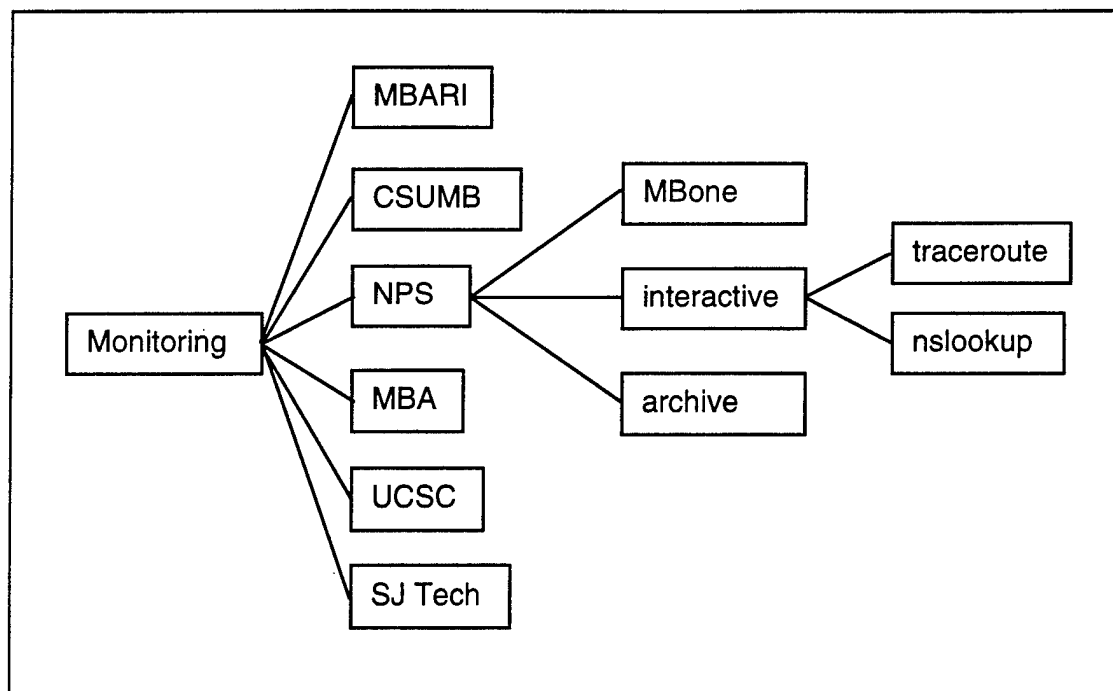


Figure 5.4. NPS Monitoring Directory Structure

***b. Language choice: perl***

Language selection for monitoring scripts is based on commonality, portability and ease. *perl* satisfies these requirements. *perl* is public-domain software that has widespread use and cross-platform compatibility. It is an interpreted language which greatly speeds debugging and modification time. It is unhindered by difficult type casting requirements and compiler setup. Text parsing, file handling and input/output are major strengths of *perl*. It is a fast and highly capable language for which plenty of documentation is available.

***c. Automated Status Monitoring***

Network status display must be easily viewed and understood to make it useful for the majority of users. Certain users desire to know more about the particular

condition and the manner in which status was attained therefore links to other sources are provided. To meet the two goals of easy access and commonality, HTTP is the protocol of choice. The top page of the BayNet monitoring suite is a clickable map. The user need not have a graphic display as the links are available without the image, but the easiest access is via the image map.

The vast majority of servers are configured to handle image mapping. Image maps are easy to use and require little administration or training. The BayNet clickable map (Figure 5.5) is the top level of the BayNet monitoring interface. All of the information available about BayNet resides at layers below the image map. The clickable map makes the logical topology readily apparent. Access to further information on any domain requires only a single mouse click. This clickable map provides "one-stop shopping" for BayNet monitoring. A single click on the domain in which the user is interested will lead to the current status of the network and an interactive interface as well as any relevant administrative facts.

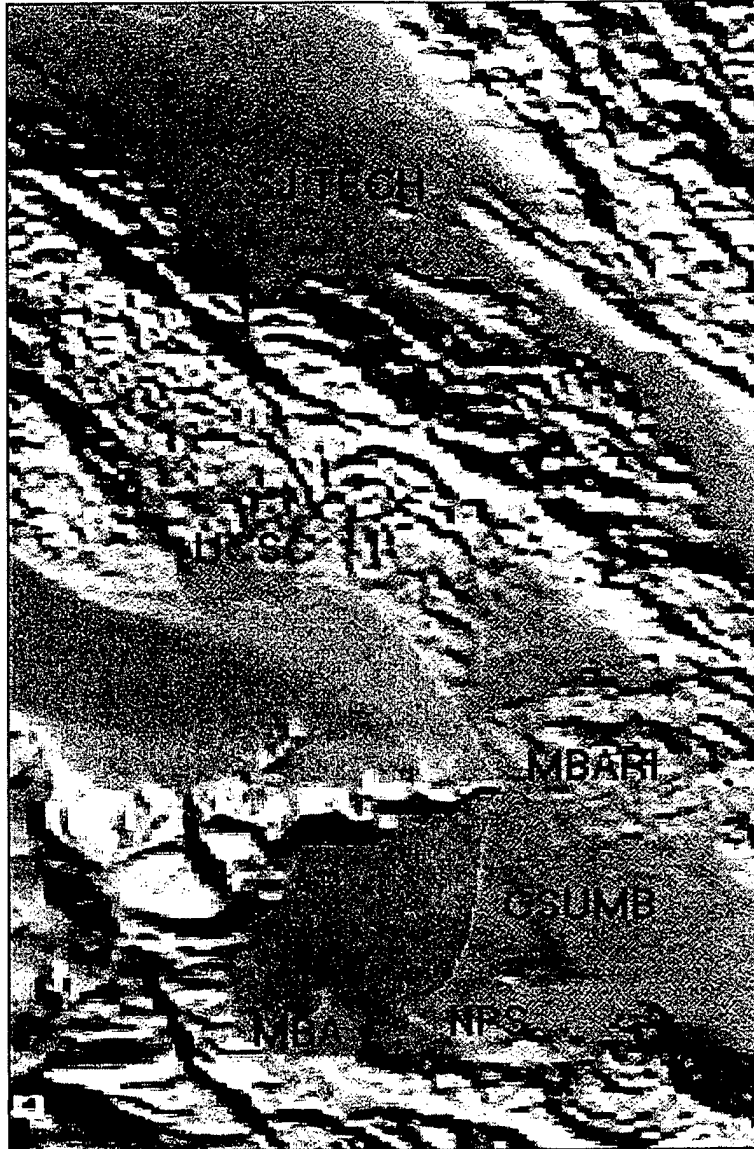


Figure 5.5 BayNet Image Map

The NPS introductory page is shown in Figure 5.6. Users can view the results of automated monitoring or test network hosts interactively real-time. The page is also a hub of data that may be needed in a network contingency or troubleshooting effort. Jumping off points include links to the scripts that are in operation, NPS-specific system details, and points of contact with phone numbers. The NPS topology can be chosen for display as can relevant work on the NPS ATM LAN. Other BayNet sites selected here will lead the user to a copy of the corresponding monitoring page with all the relevant information for that site.

The Universal Resource Locator (URL) for this page is  
[http://www.stl.nps.navy.mil/~iirg/atm/monitoring/Ping\\_pages/NPS/monitoring.html](http://www.stl.nps.navy.mil/~iirg/atm/monitoring/Ping_pages/NPS/monitoring.html)

---

#### NPS Monitoring Page

---

##### **NPS Monitoring page**

- o Displays the current (half-hourly) status of the Naval Postgraduate School's ATM network. Currently running from steel
- o *perl* monitoring script
- o *crontab* command

##### **NPS Interactive Monitoring Page**

- o Interactively test hosts on the Naval Postgraduate School's ATM network.
  - o *ping* script
  - o *traceroute* script
  - o *nslookup* script
- 

##### **NPS Topology**

##### **NPS ATM LAN**

---

NPS ATM network *ping* project home page: (edwardse@nps.navy.mil)  
Last update: 20 May, 1996

---

|BayNet *ping* page|

|UC Santa Cruz *ping* page|

|MBARI *ping* page|

|Monterey Bay Aquarium *ping* page|

Figure 5.6. NPS monitoring page

Selection of the NPS monitoring page displays the results of automated testing. This display begins with the originating host's address and the timestamp. It is organized alphabetically by domain. Each domain is given as a bold title with a link to the hosts that were tested. Which hosts that are testable can be easily verified again by a single mouse click. The `hosts.txt` files only contain the host names or IP numbers, which match the format of standard file `/etc/hosts` as in the following example from the NPS Computer Science network. This greatly simplifies alteration procedures.

```
% more hosts.txt.cs  
cs.nps.navy.mil  
meatloaf.cs.nps.navy.mil
```

Most users at NPS are familiar with the Computer Science network and its main hosts. Other networks or portions of networks may not be so well known. In this scenario, additional information can augment the host file. NPS building 230, the golf course lab is a remote site connected to the Computer Science LAN via a wireless bridge. Most network users are unfamiliar with the golf course hosts. Ordinary problems involving golf course hosts may not be easily remedied due to this unfamiliarity. As can be seen in the golf course test file, several lines of amplifying information accompany the hosts data. The comments available on the monitoring page greatly simplify



troubleshooting network problems involving golfcourse hosts. Comments include points of contact, phone numbers, areas of responsibility and IP numbers.

```
% more hosts.txt.golfcourse
# AUV golf course lab
# reserved IP Nos: 131.120.7.100 - 119
# golf course lab    Bldg 230                656.2438
# graphics lab      Span 506                656.2037
# Sue Whalen        whalen@cs.nps.navy.mil    656.2967
# Rosalie Johnson   johnsonr@cs.nps.navy.mil    656.3392
# Russ Whalen       whalenr@cs.nps.navy.mil    656.2093
# wireless bridge at graphics net in Spanagel
131.120.7.100      airlan1.cs.nps.navy.mil    airlan
# wireless bridge Bldg 230 lab at the golf course
131.120.7.101      airlan2.cs.nps.navy.mil    airlan2 airlan2-gc
131.120.7.105      indy-gc.cs.nps.navy.mil     indy-gc
```

Thus additional information is only displayed where absolutely necessary.

This presents a clean clear display. The host text files that have amplifying information in them will display it each time the page is generated. The host files are essentially small databases that can change independently of the means of testing them. Unlike many databases, the host files are easily accessed because it is important for users to know what hosts are being tested. The 'hosts tested' link will lead to the actual file that contains the host information for that particular domain. Figure 5.7 shows mixed results; some have comments, some don't. The behavior of the network will dictate to some degree what comments are useful (e.g., hosts that crash on a regular basis require augmenting

comments to track performance). Additionally, the 'alive' hosts are presented in green and the 'down' hosts in flashing bold red. This alleviates the necessity to read each line. The points of interest are readily apparent.

```
NPS Monitoring Page
-----
Ping from NPS to various BayNet sites
Mon Jul 22 11:00:01 PDT 1996
-----
golcourse
hosts tested
-----
# AUV golf course lab
# reserved IP numbers: 131.120.7.100 thru 131.120.7.119
# golf course lab      Building 230      656.2438
# graphics lab        Spanagel 506      656.2037
# Sue Whalen          whalen@cs.nps.navy.mil    656.2967
# Rosalie Johnson     johnsonr@cs.nps.navy.mil    656.3392
# wireless bridge at graphics net in Spanagel
131.120.7.100 is alive
# wireless bridge at building 230 lab at the golf course
131.120.7.101 is alive
131.120.7.105 is alive
-----
npsfeed
hosts tested
-----
131.120.54.1 is alive
131.120.252.101 is alive
131.120.251.51 is dead
131.119.0.193 is alive
-----
ucsc
hosts tested
-----
cse.ucsc.edu is alive
terra.cse.ucsc.edu is alive
ftp.cse.ucsc.edu is alive
-----
| BayNet Monitoring Page |
| NPS Monitoring Page |
```

Figure 5.7. NPS Status

A major consideration in the design of automated monitoring is portability. BayNet composition is highly varied and complex. Differences exist in operating systems, platforms and server setup. While a version of automated monitoring may be tightly adapted to the individual network, the disadvantages of doing so outweigh the advantages. Commonality is the key to upgrades and modifications that are easily implemented. One of the recommendations for future work is to design a script which automatically polls from each of the BayNet domains back to the master version (i.e. NPS) to compare whether their version of software is up to date. If required the script will update itself.

#### *d. crontab*

The clock daemon executes commands at specified dates and times. Regularly scheduled commands can be specified according to instructions found in *crontab* files in the directory `/usr/spool/cron/crontabs`. Users can submit their own clock daemon instruction via the *crontab* command. *crontab* status can be verified by checking in the main cron directory `/usr/lib/cron`. A history of all actions taken by cron are recorded in `/usr/lib/cron/log`.

Users access the clock daemon by the *crontab* command. *crontab* copies the specified file, (or standard input if no file is specified), into a directory that holds all users' *crontabs*. The `-r` option removes a user's *crontab* from the *crontab* directory. The command `crontab -l` will list the active *crontab* jobs for the invoking user on the current host.

Not all network administrators permit wide use of *crontab*. Users may implement a 'cronjob' if their names appear in the file `/usr/lib/cron/cron.allow`. If that file does not exist, the file `/usr/lib/cron/cron.deny` is checked to determine if the user should be denied access to *crontab*. If neither file exists, only root is allowed to submit a job. If `cron.allow` does not exist and `cron.deny` exists but is empty, global usage is permitted.

The best way to run *crontab* jobs is to house the *crontab* information in a one-line test file. In this case the file must be only one line of data with the fields separated by a tab or space. The command to initiate *crontab* in this case is '`crontab cronjob`' where `cronjob` is the file holding the execution timing information. This presents a more elegant solution than stopping and starting the *crontab* each time a change is made.

The arguments to *crontab* whether in a text file or at the command line are six fields separated by spaces or tabs. The first five are integer patterns as specified below, the sixth is the name of an executable file.

minute (0-59)

hour (0-23)

day of the month (1-31)

month of the year (1-12)

day of the week (0-6 with 0=Sunday)

Each of these patterns may be either an asterisk (meaning all legal values) or a list of elements separated by commas. An element is either a single integer, multiple integers separated by commas, or two integers separated by a minus sign (meaning an inclusive range). The specification of days may be made by two fields (day of the month and day of the week). If both are specified as a list of elements, both are adhered to.

This *crontab* runs program.exe on the first and fifteenth of each month, as well as on every Monday. To specify days by only one field, the other field should be set to \*, (for example, 0 0 \* \* 1 would run a command only on Mondays):

```
0    0    1,15    *    1    program.exe
```

In this example program.exe runs the executable at minute 00 and minute 30 of every hour every day of the year:

```
0,30 *    *    *    *    program.exe
```

*crontab*'s behavior is unpredictable when calling across directories. When calling a simple program, surprises are uncommon. When an program in a directory other than the initiator's home directory is called, or when called from a remote directory, inconsistencies appear especially when the executable uses other files. In these situations, care must be taken to get the proper results. A common problem is that file visibility failures can not be duplicated. This may be due to the fact that *crontab* uses the default shell (sh) rather than the users shell, so path information is not available.

Consider the example above and with the addition that it writes HTML files to and from the same directory it which it resides. To start this *crontab* from the IIRG directory looks like this:

```
0,30 * * * * * program.exe
```

This *crontab* will run provided that *program.exe* is in the *~iirg* publicly visible directory. Any HTML files that the program accesses must also be in the local *~iirg/.public\_html* directory. *crontab* failures are either redirected or mailed to the initiator. Since it is very useful to know when *crontab* goes awry, the mail option is commonly used. When using a group account such as *~iirg*, *crontab* failure mail goes to every member of the group. This is not a favorable outcome. Thus it is desirable to start this *crontab* from the single responsible user's account.

When the following command is begun from *~edwardse* account a few problems arise.

```
0,30 * * * * * ~iirg/program.exe
```

Now the e-mail in the event of failure will go to *edwardse*. But the HTML files upon which the program depends are no longer taken from *~iirg/.public\_html* but from *edwardse/.public\_html*. The files can be linked but this is not practicable when creating and destroying files automatically. To overcome this problem requires explicit instructions to the *crontab* command. These instructions can be changes of directory, remote shell execution path or others as required. Note that all directory changes must be fully specified since *crontab* does not use default path information.

```
0,20,40 * * * * * cd home/faculty/  
iirg/.public_html/atm/monitoring/NPS;  
perl program.exe
```

This *crontab* command is wrapped around for the sake of printing, but in actuality it is a single long line. *crontab* commands must be a single line entry when initiated. The command, initiated from the user account `~edwardse`, runs the NPS automated monitoring script which is located in the group account `~iirg/.public_html/atm/monitoring/NPS`; at minutes 00 , 20 and 40, every hour, every day of the month, every month of the year, seven days a week. When there is a problem with the command for some reason, the notification is sent to the invoking user (`edwardse`).

*crontab* commands execute on the machine where they are started. So to verify a current cronjob, one must know on which machine it resides. This may cause some embarrassment after a *crontab* script has been running for some time and the initiator has forgotten where the process began. One solution to this is the '*rshstl*' program. The code for *rshstl* is included as Appendix L. *rshstl* runs a user-specified command against all STL hosts. This simple *perl* program performs a 'rsh' remote shell command, executed remotely on each STL host. *rshstl* can easily be modified for other LANs by replacing host names.

The *rshstl* program tests the hosts that are loaded in the `@hosts` array against the command `crontab -l`. The standard reply of *crontab* when there are no *crontab* scripts designated for use on the particular host is

```
crontab: can't open your crontab file.
```

If a host does have a resident *crontab* command loaded under the users cron log, it will reply with the specific *crontab* command. The example results that follow show that the crontab is running on baby.

```
%rshstl
```

```
rsh azure.stl.nps.navy.mil crontab -l  
crontab: can't open your crontab file.
```

```
rsh baby.stl.nps.navy.mil crontab -l  
0,10,20,30,40,50 * * * * * ping_host.pl
```

```
rsh blackand.stl.nps.navy.mil crontab -l  
crontab: can't open your crontab file.
```

Before assigning an executable file to a *crontab* operation, it must be thoroughly tested. The best way to test the *crontab* implementation is to put it in the intended directory and limit any dependent file and data fields to the smallest possible. Run it repeatedly from the command line as on initial test. Then set the *crontab* to call the program every minute: \* \* \* \* \* program.exe. This allows for steady feedback. When there are no bugs then the directory structure can be implemented. Finally when the executable, the associated files and the e-mail performs as expected, change the *crontab* to the interval desired for normal operation. Adjust any prerequisite



data such as testable hosts to their real-world values. Hopefully the debugging effort will be over at this point.

#### *e. Saving State*

Since monitoring is best accomplished without the use of continuously executing programs, saving and retrieving program state information between invocations becomes a major consideration. The state of various hosts, syntax and format requirements, and system requirements can be stored as local files, using either file name or file contents to save important state information. *perl* is ideally suited to state parameter saving using file storage because of its inherently powerful file handling input/output capabilities. Since monitoring functionality does not require non-stop high-performance operation, file operations using *perl* provides all the state 'memory' functionality necessary by using file creation and destruction and appending. Additionally, automated file construction and destruction enhances e-mail reporting and archival analysis capabilities. The full *perl* code for automatic status monitoring is included as Appendix M.

### **3. Interactive Monitoring**

From the NPS Monitoring page, the **NPS Interactive Monitoring Page** link will take the user to the following page shown as Figure 5.8.

Interactive CGI

url for this page is:

<http://blackand.stl.nps.navy.mil/~iirg/atm/monitoring/NPS/interactive/interactive.cgi>

Other Tools

**traceroute** NPS and other BayNet hosts

**nslookup** NPS and other BayNet hosts

Choose one or more of these hosts to *ping*

The test will be performed from blackand.stl.nps.navy.mil

cc

- ☐ nps.navy.mil
- ☐ work2.cc.nps.navy.mil
- ☐ amadeus.as.nps.navy.mil

cs

- ☐ cs.nps.navy.mil
- ☐ meatloaf.cs.nps.navy.mil

npsfeed

- ☐ 131.120.54.1
- ☐ 131.120.254.20
- ☐ 131.120.252.101
- ☐ 131.120.251.51
- ☐ 131.119.2.53
- ☐ 131.119.0.193

stl

- ☐ blackand.stl.nps.navy.mil
- ☐ royal.stl.nps.navy.mil
- ☐ navy.stl.nps.navy.mil
- ☐ azure.stl.nps.navy.mil
- ☐ cadet.stl.nps.navy.mil
- ☐ steel.stl.nps.navy.mil
- ☐ spot.stl.nps.navy.mil
- ☐ slate.stl.nps.navy.mil
- ☐ fletch.stl.nps.navy.mil
- ☐ cirrus.stl.nps.navy.mil

ucsc

- ☐ cse.ucsc.edu
- ☐ terra.cse.ucsc.edu
- ☐ ftp.cse.ucsc.edu

☐ PING HOSTS

Figure 5.8. NPS ping page

The interactive monitoring of the network contains three of the most useful tools to determine network status: *ping*, *traceroute* and *nslookup*. Since *ping* is the most common start to a troubleshooting effort, it is immediately accessible on the interactive page. *traceroute* and *nslookup* are selectable links. The interactive testing capability is designed to provide nearly instantaneous feedback from a number of network monitoring tools without command line interaction.

The hosts that are selectable on all of the interactive tests are configured in real time using host files. Thus the code is totally separate from the data. Host status can change without corresponding code revisions. The code will react to the host changes in the subsequent execution.

The code for the interactive tests are comboform CGI scripts. [Brenner, Aoki, 1996] Comboform refers to a single script capability that combines a default query form with the ability to answer form-generated queries. They are a sophisticated and powerful form of scripting in which a single script displays the initial form in HTML, processes the input (if any), then displays the appropriate results. The advantages of this method are that there are less program files to keep current and working. There are no additional server requirements to support CGI comboforms since they are merely self-querying scripts.

Command line use of *ping* is fairly simple although not without pitfalls. In comparison, *traceroute* is a little more difficult and *nslookup* is intimidating. Minimally speaking, the proper use of all these tools requires the ability to read and understand the

man pages and possibly some hands-on training. By designing ease of use through comboform scripting, the decisions that these tests demand are shifted to the writer of the script rather than the user. The choice of the command-line switches to use, the hosts that are available to test and the display of the results are controlled by a carefully constructed and validated script rather than the user.

*ping* switches are numerous and powerful. However the switch syntax varies among operating systems, a common occurrence even within a small LAN. *ping* can cause severe network degradation in certain circumstances such as an unconstrained flood *ping*. The responsibility for correct switch selection has traditionally rested with the user, usually requiring heavy man page interaction. Since scripted testing moves this responsibility from the user to the script, the format of the *ping* can be correct for each test. This form of interaction is both safe efficient. Consider an internal test (within a LAN) of four hosts . The manual method requires the four separate command line interactions:

```
% ping cadet
```

```
cadet.stl.nps.navy.mil is alive
```

```
% ping royal
```

```
royal.stl.nps.navy.mil is alive
```

```
% ping steel
```

```
steel.stl.nps.navy.mil is alive
```

```
% ping navy
```

```
navy.stl.nps.navy.mil is alive
```

This test required over 30 seconds.

The same test performed by a CGI script requires the user to simply select the four hosts from this menu shown in Figure 5.9. Prior knowledge of the full name of the hosts (or even what hosts are available) is not required.

Choose one or more of these hosts to *ping*  
The test will be performed from blackand.stl.nps.navy.mil

---

stl

<input type="checkbox"/>	blackand.stl.nps.navy.mil
<input checked="" type="checkbox"/>	royal.stl.nps.navy.mil
<input checked="" type="checkbox"/>	navy.stl.nps.navy.mil
<input type="checkbox"/>	azure.stl.nps.navy.mil
<input checked="" type="checkbox"/>	cadet.stl.nps.navy.mil
<input checked="" type="checkbox"/>	steel.stl.nps.navy.mil
<input type="checkbox"/>	spot.stl.nps.navy.mil
<input type="checkbox"/>	slate.stl.nps.navy.mil
<input type="checkbox"/>	fletch.stl.nps.navy.mil
<input type="checkbox"/>	cirrus.stl.nps.navy.mil

---

☐ ping hosts

Figure 5.9. Completed interactive *ping* selection

Figure 5.10 is the Netscape display of the results. From the time the interactive page was displayed on the browser until the time the results were displayed took under five seconds. The interactive *ping* code is included as Appendix N.

Interactive Network Monitoring from  
blackand.stl.nps.navy.mil  
Tue Jul 23 11:46:45 PDT 1996

---

ping to royal.stl.nps.navy.mil

Command Line equivalent: *ping -c 1 royal.stl.nps.navy.mil*

**royal.stl.nps.navy.mil is alive**

ping to navy.stl.nps.navy.mil

Command Line equivalent: *ping -c 1 navy.stl.nps.navy.mil*

**navy.stl.nps.navy.mil is alive**

ping to cadet.stl.nps.navy.mil

Command Line equivalent: *ping -c 1 cadet.stl.nps.navy.mil*

**cadet.stl.nps.navy.mil is alive**

ping to steel.stl.nps.navy.mil

Command Line equivalent: *ping -c 1 steel.stl.nps.navy.mil*

**steel.stl.nps.navy.mil is alive**

---

| ping | traceroute | nslookup |

Figure 5.10. *ping* results

*traceroute* is selected in exactly the same way and the results are given in the same format. Similar efficiency improvements exist. The CGI script for *traceroute* further enhances the test by providing the user with the hosts that are available to analyze. A common stumbling block in the analysis of traffic between LANs is an unfamiliarity with the remote LAN. Scripted *traceroute* overcomes this problem because the hosts are listed. For example, if a user wants to trace a packet to UCSC, the interactive *traceroute* display lists that for him. This alleviates the need to ascertain and manually enter an unfamiliar domain name. It also allows for a quick and easy selection of other hosts at that site in the event of a failed trace. In fact the entire network may be traced as desired.

*traceroute* from the command line is often not a simple procedure. Because it is sometimes difficult, it is under utilized. Scripting has therefore turned *traceroute* into an easy to use and highly beneficial tool for any user. The code for `traceroute.cgi` is included as Appendix O.

Typically *nslookup* is even more difficult to use than *traceroute*. The command-line test comes in two forms: interactive and non-interactive. *nslookup* demands that the user know the exact destination domain name or IP address. *nslookup* has numerous switches and combinations of switches, host names, file names and redirection. The following interactive *nslookup* illustrates its usefulness.

```
% nslookup
Default Server:  azure-63.stl.nps.navy.mil
Address:  131.120.63.1
```

```
> ls -a stl.nps.navy.mil
[azure-63.stl.nps.navy.mil]
loghost          azure.stl.nps.navy.mil
kerberos         spot.stl.nps.navy.mil
gate-azure       azure-63.stl.nps.navy.mil
periwinkle       blackand.stl.nps.navy.mil
www              azure.stl.nps.navy.mil
ftp              azure.stl.nps.navy.mil
```

As users become more and more adept in managing their own network accounts and applications, the more beneficial this type of information becomes. Most users do not attempt *nslookup* (especially the interactive version) because it is awkward. Scripting a well designed HTML page interface eliminates this awkwardness. The code for `nslookup.cgi` is included as Appendix P.

```
url is http://blackand.stl.nps.navy.mil/~iirg/atm/monitoring/
NPS/interactive/nslookup/nslookup.cgi

-----

This is an interactive program to query Internet domain name servers.
The test will be performed from blackand.stl.nps.navy.mil

-----

Choose a host - server info will be returned
☐ System Technology Lab
☐ Computer Science
☐ Computer Center

-----

☐ perform nslookup
```

Figure 5.11. Interactive nslookup



Figure 5.11 shows the interactive version of *nslookup*. The interactive command for each host is stored in a file in the *nslookup* directory. When the `nslookup.cgi` executes, it does not begin *nslookup* until it reads the appropriate file; i.e. the file that holds the command for the domain or domains that were selected. For the Computer Science department, the text file `nslookup.txt.cs.nps.navy.mil` holds the following interactive portion of the *nslookup* command:

```
% more nslookup.txt.cs.nps.navy.mil
ls -a cs.nps.navy.mil
exit
```

This is the method by which the CGI comboform can execute the interactive *nslookup* command. The result as shown in Figure 5.12 is a listing of the servers on the Computer Science department network. This is extremely useful information, not just in troubleshooting but in network familiarity in general. All users benefit from having this information at their fingertips. The instructions for installation of the monitoring software are included as Appendix Q.

Interactive version nslookup  
Tue Jul 23 12:31:02 PDT 1996

---

Default Server: cs.nps.navy.mil

Address: 131.120.1.13

> [cs.nps.navy.mil]

annex1	annexbox1.cs.nps.navy.mil
annex2	annexbox2.cs.nps.navy.mil
wwwcaps	suns5.cs.nps.navy.mil
sgihc	caesar.cs.nps.navy.mil
ftp	cs.nps.navy.mil
taurus	cs.nps.navy.mil
gravy7	meatloaf.cs.nps.navy.mil
arlan1	arlan_shore.cs.nps.navy.mil
arlan2	arlan_auv.cs.nps.navy.mil
hubwatch	pcstaff.cs.nps.navy.mil
www	cs.nps.navy.mil

---

| Interactive Test | nslookup | traceroute |

Figure 5.12. *nslookup* results

## **G. BAYNET ATM**

The focus of BayNet ATM is on educational and research applications. Baylink is the primary application that runs on BayNet ATM. Baylink is an ATM connection jointing MBARI, MBA, and SJTMI with the MBARI research vessels underway in the Monterey Bay Canyon. UCSC and their extension have a dedicated multiconferencing system for distance learning which utilizes the BayNet ATM connection. A number of other applications have been proposed for BayNet but have not progressed beyond the formative stages.

The North-South connection is a DS-3 backbone between LATAs 7 and 8 which can deliver 45 Mbps. Unfortunately, the Sprint switch which is in the middle of BayNet ATM cannot be configured to run SVCs despite two years of requests thus any traffic between the northern and southern LATAs must be configured using PVCs. This simplifies the monitoring aspect of the BayNet ATM connection but severely decreases the advantage of ATM over BayNet since PVCs require a fixed bandwidth allocation. A detailed analysis of BayNet ATM appears in [Courtney, 96], including further explanations of the following major shortcomings of ATM.

### **1. Interoperability**

A significant hindrance to the establishment, implementation and monitoring effort of BayNet has been a mismatch of switch hardware and software. The NPS ATM LAN consists of three Cisco switches and 20 Fore card adapters. The other regional partners have mostly Fore switches and software. The Sprint switch at the center of

BayNet is Newbridge. Partners other than NPS utilize proprietary hardware and software, requiring large expenditures and diverting effort from open solutions (such as the MBone tools). Lack of standards and proprietary dissimilarity have caused numerous delays and (in the end) lack of performance.

## **2. Incompatibility**

ATM and IP must exist together. They do not in the BayNet configuration co-exist in a proficient fashion. The problems revolve around addressing issues and the lack of standards, both of which have already been covered. No clear solution has yet been proposed that maps connection-oriented ATM with many-to-many Multicast IP.

## **3. Inflexibility and Tariffs**

Because BayNet involves numerous entities, each iteration of the topology, the switching configuration, or the bandwidth allocation requires a general agreement of several people. Manual intervention by PacBell follows consensus, causing further delays. These problems are magnified by entry-level costs sums to tens of thousands of dollars when attempting cross-country connections.

## **4. BayNet Monitoring**

The intention of BayNet monitoring is to prove the portability of the tools and regional scale monitoring. The challenge of BayNet monitoring is to have the capability to achieve the monitoring objectives on various platforms and operating systems. This thesis has shown that these goals are achievable using IP.

## **5. Future of BayNet**

The BayNet ATM regional partnership is disintegrating because of the looming cost considerations and the lack of dependable connectivity. To continue the 48 Mbps DS-3 connection will cost each user over 5,000 dollars per month. A 155 Mbps OC-3c is almost 8,000 dollars per month. There are currently no applications running over BayNet link that justify that expenditure. The failures of this project occurred despite massive expenditures of time, effort and resources. We remain hopeful that ATM can overcome these significant problems but more work remains to be done. [Courtney, 96]

## **H. SUMMARY**

There is no doubt that monitoring is crucial to all networks. It is vital to the individual concerns of commercial, military and academic network endeavors and also to the health of the Internet as a whole. Public-domain software applications are a key component of network monitoring efforts. Address resolution is a difficult problem at times but certainly not insurmountable. Topology can become familiar by having the ability to determine it readily and often. Automated and interactive monitoring have been integrated without significant cost to network performance, storage requirements or administrative burden. The monitoring package was easily ported to BayNet domains without major effort thus demonstrating an ability to perform regional monitoring using public-domain software, CGI scripts and HTML interfaces.



## **VI. SECURITY ASPECTS OF CONTINUOUS NETWORK MONITORING**

### **A. INTRODUCTION**

Network monitoring plays an influential role in network security. Network status and fault location are of vast importance. Network security typically suggests intrusion protection against "hacker" mechanisms, worms or viruses. This definition excludes the equally important aspect of physical network security. The ultimate denial of service is the malicious removal or damage of equipment. Considered from this perspective, network monitoring can deter physical intrusion as well as provide substantial information in the event of a violation.

### **B. DESIGN CRITERIA**

The design of a network monitoring system for physical security and denial-of-service detection must meet two criteria: it must be continuous and it must use existing hardware and software. It is absolutely essential that the system function 24 hours a day, 7 days a week, 365 days a year. A gap in coverage represents a serious security breach which nullifies the overall integrity of the system. The rate of network testing must be as high as practical without overloading system resources. It certainly must be less than the period of time in which reaction to a compromised network needs to occur. Finally, the system must not be shackled to proprietary hardware and software.

## **C. PURPOSE**

This project, referred to here as the Continuous Site Monitoring System (CSMS) is designed to fulfill two major goals: monitoring and warning.

### **1. Continuous Site Monitoring**

A designated site (presumed to be of high value), might be monitored using MBone audio and video recording (MBone VCR). For full coverage of MBone VCR see [Tiddy, 96]. Conceptually, the MBone video archive will operate in a manner similar to commercial video surveillance which continuously rerecords the same media over a continuous cycle. The MBone VCR application can record MBone audio and video to network storage located geographically distant from the monitored site. At given intervals the recording will be saved and timestamped. After a comfortable period of time has elapsed without incident, the archives can be overwritten in sequential order. If qualifying factors justify the expense, the MBone archive can be automatically renamed and permanently stored.

### **2. Status Warning**

Certain factors will trigger the emergency response mechanism of the system. The main reason is if a host becomes inactive. The testing time interval can be changed easily via the software. If it is assumed that maximum protection is desired, a feasible default interval is one minute. In the event of host status change, the designated administrator can be notified via e-mail to an alphanumeric pager. The message will carry a brief description of the events such as test time, host name and last positive test.



Two hosts not located within the same network will verify the host status. In the event of a genuine high-value host failure, two e-mail notifications will be posted. Once a host has been reported inactive, the status will continue to be recorded every monitoring interval but not e-mailed.

When a status warning has occurred, the software will archive all available MBone VCR data. The event will carry the unique identifier of its timestamp. Figure 6.1 demonstrates the basic configuration of the design.

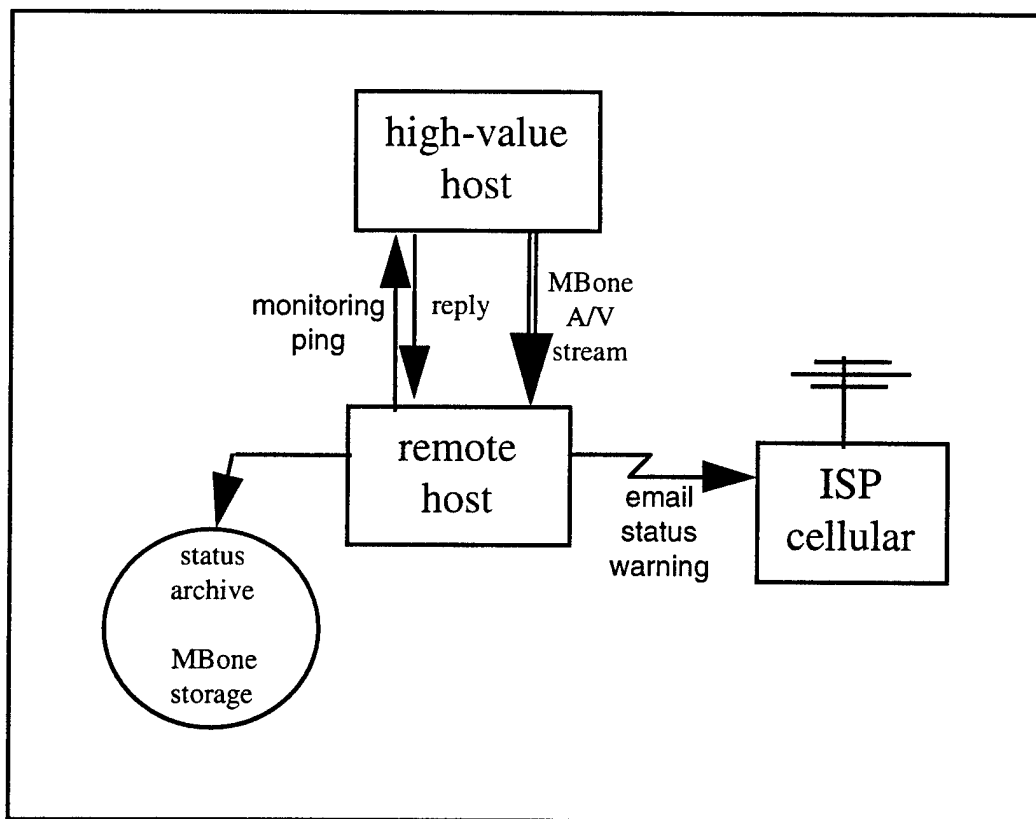


Figure 6.1. Basic Configuration

## D. METHODOLOGY

### 1. NPS Golf Course Lab

A remote site that is part of the NPS Computer Science department LAN is the test bed for CSMS. It is in use as a testing facility for autonomous underwater vehicles (AUVs). The facility is generally unmanned at night but its network remains active.

The golf course lab has a transceiver for the airlan wireless bridge operating three miles from the NPS campus. The NPS *Phoenix* AUV and test tank are located there as well as associated support equipment. *sd* software operating from a Silicon Graphics Indigo workstation with IndyCam provide MBone audio/visual signal.

### 2. Monitoring Algorithms

The monitored host, if disconnected, will be unable to provide any relevant information. So the host will be tested from two other sites, one on the same network and one elsewhere. Where the *ping* originates is extremely important.

If it originates on the same subnet then there are no points of failure between the two hosts. Thus a failure constitutes either a failure of the remote host to reply to a *ping* or a failure of the *pinging* host. The *ping* will be initiated via a *crontab* script on the *pinging* host. Since the *crontab* is run from the *pinging* host, if that host goes down there will not be *crontab* generated failure indications. For *pings* that begin on another network or subnet, other points of failure are introduced. Because the remote host is monitored from two separate points, a total monitoring failure will only occur in the event of failures on two independent subnets. Such an arrangement is a good candidate for

future work building a simple rule-based expert system that can prioritize faults based on network topology.

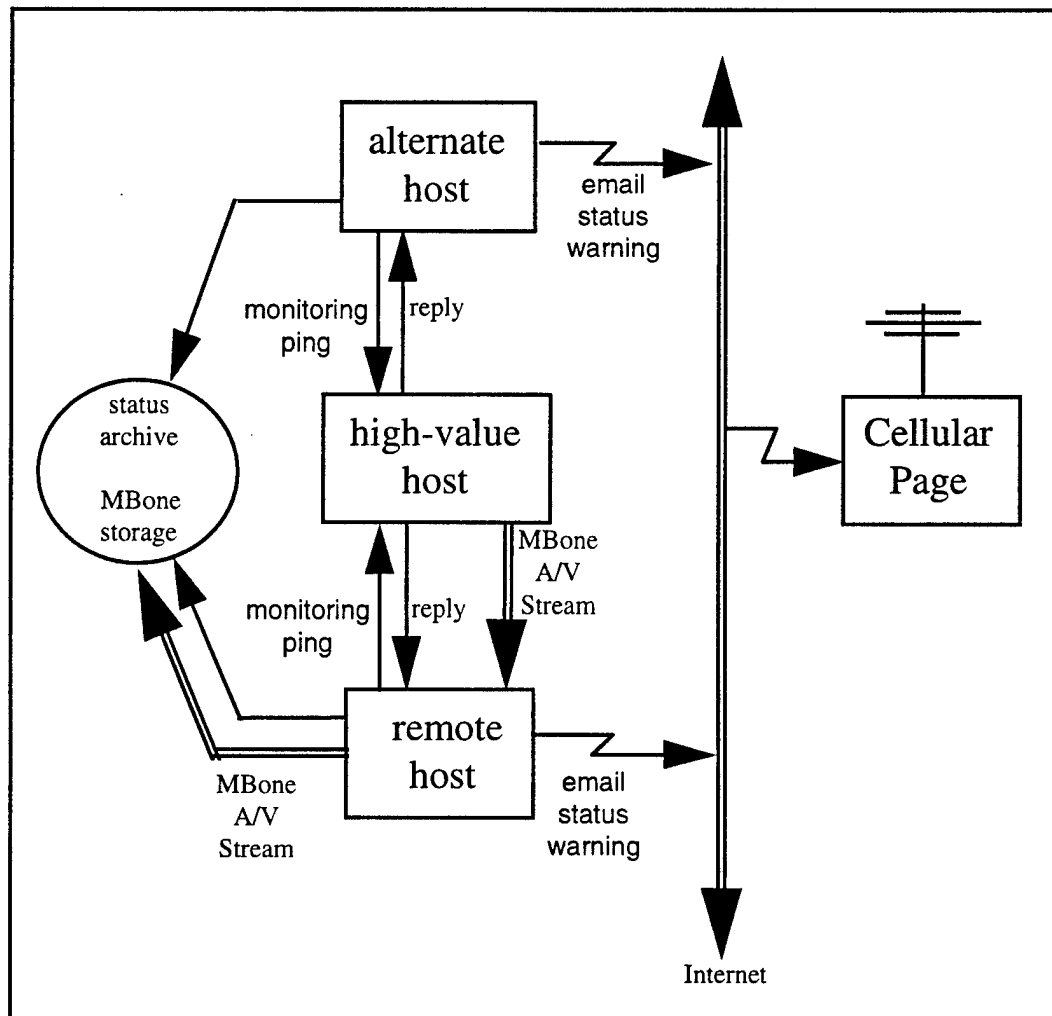


Figure 6.2. Network Independent Configuration with redundancy

As shown in Figure 6.2, redundancy is valuable in the testing of the host, the archival of Mbone VCR, and in the notification of the administrator. When host status becomes inactive, notification is immediately accomplished via two separate e-mail

messages to the administrator's alphanumeric pager. At the same time, MBone VCR is archived. A bonafide inactive status is therefore comprised of two separate (but nearly simultaneous) alphanumeric pager notifications.

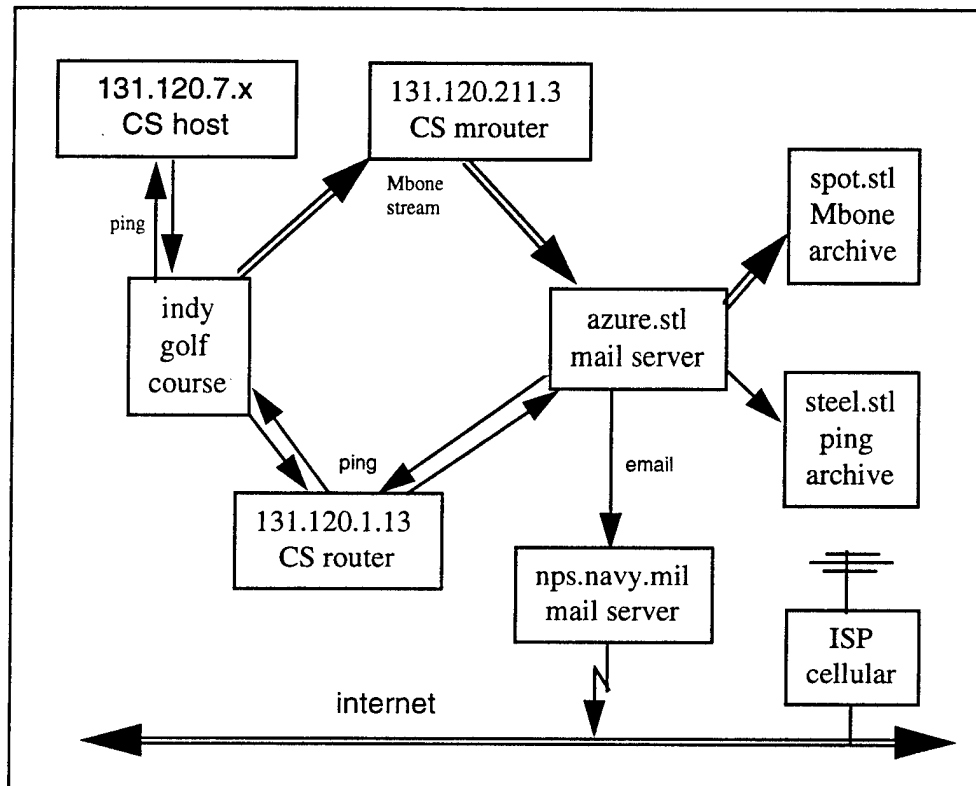


Figure 6.3. NPS Configuration

As shown in the Figure 6.3, NPS does not have a subnet host with it's own Internet access and mail server. This is a weakness of the system at NPS. All incoming and outgoing campus mail is routed through the Computer Center. This is a series of single points of failure. There are at minimum three points of failure other than the remote host itself because of the manner in which e-mail is controlled; taurus.cs.nps.navy.mil (CS LAN mail server), azure.stl.nps.navy.mil (STL mail server)

and nps.navy.mil (NPS mail server). Incorporating Internet access on the subnet along with a private mail server is the ideal setup for the emergency notification activation.

*a. ping failure from within the same subnet*

The best way to minimize points of failure is to run a *crontab* script that *pings* the remote host from the server on the same subnet. In Figure 6.4 the *crontab* script is run from another host on the subnet. A failed test from the high value host will be discovered by the other remote host then passed through the subnet server. The archive procedure and the e-mail notification must then travel through an additional host (CS router).

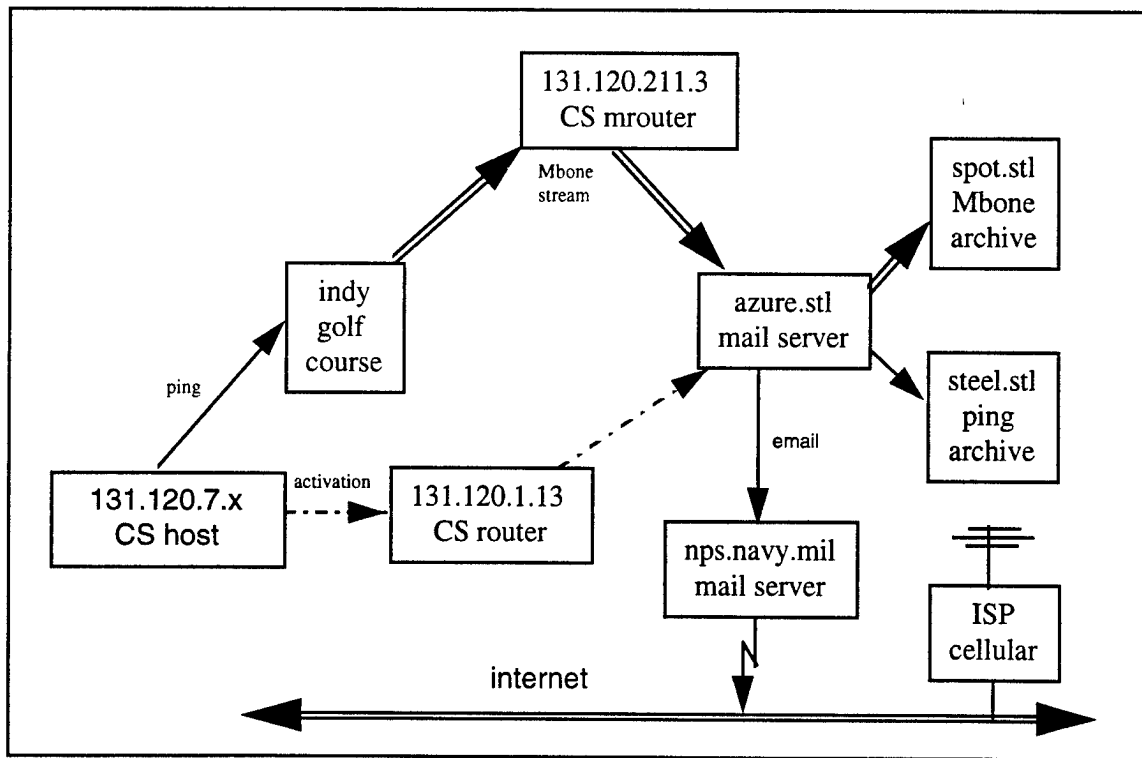


Figure 6.4. Failed *ping* response

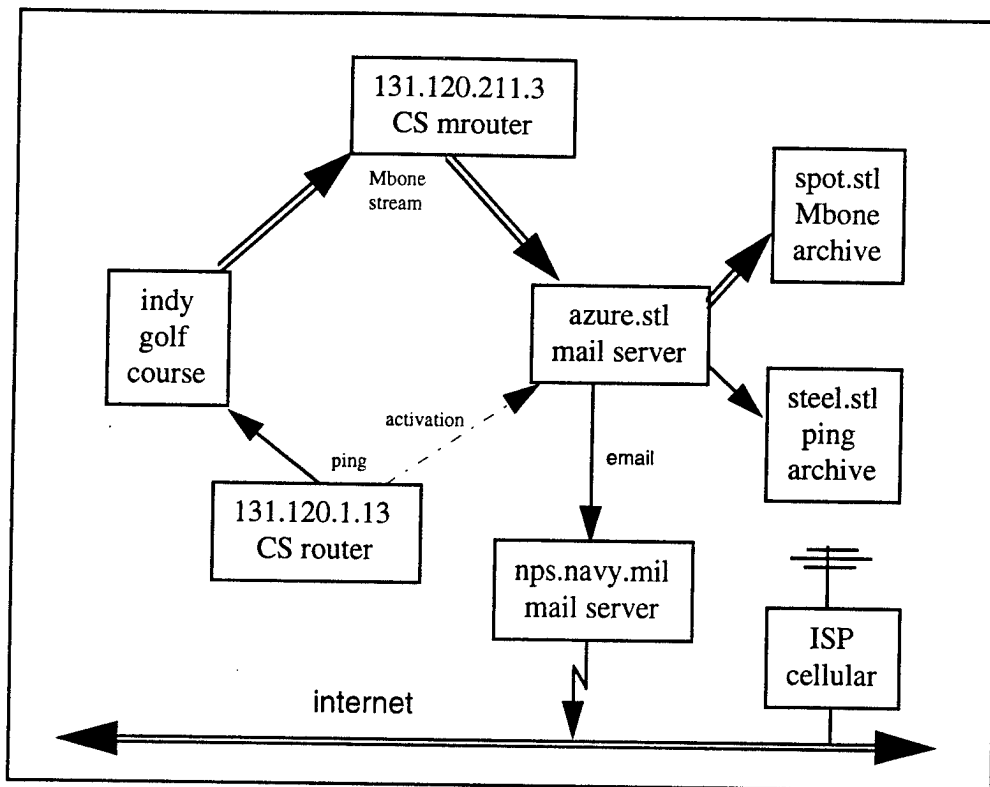


Figure 6.5. Failed *ping* from subnet server

In Figure 6.5, the subnet server is the *pinging* host thereby lessening the intermediate points of failure. The CS router does not receive a response from the *ping* and thus begins the activation procedure. Servers and routers tend to function at higher loading than other network hosts. Because of this servers may not be the correct choice for remote testing in all networks.

***b. high-value host verification from a different subnet***

*ping* from a different network or subnet introduces potential gaps in the monitoring of the remote host. Two failures are possible. First is an actual failure of one or both of the *pinging* hosts, second is the failure of the notification procedure. In the event of a failure of a *pinging* host, coverage is provided by the other *pinging* host. The

failure of the notification via e-mail is a genuine single point of failure. If true redundancy is desired, two separate e-mail mechanisms must be in place. Remote host monitoring is not totally incapacitated in the event of a e-mail related casualty such as a mail server failure, it is simply without redundancy. Figure 6.6 demonstrates the result of a controlled contingency.

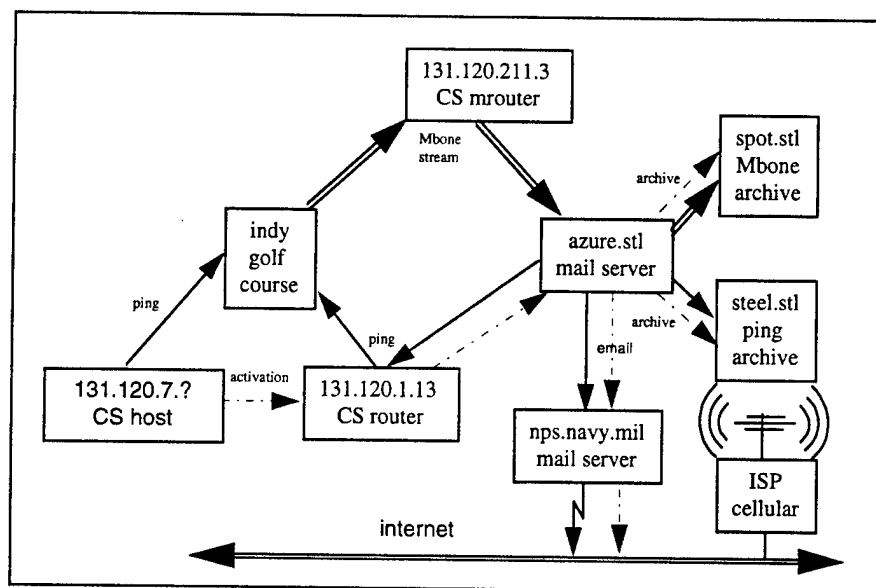


Figure 6.6. Contingency Operation

Although these scenarios are somewhat involved, they hold great promise. Day-to-day monitoring eliminates bugs and minimizes spurious false alarms. We expect that it is only a matter of time before this approach provides rapid notification of an actual physical security problem such as theft or fire in progress

## E. E-MAIL PAGER NOTIFICATION

The e-mail pager service requires two prerequisites, an alphanumeric pager with cellular paging service, and an Internet service account configured to forward e-mail to

the pager. The amount of information is limited by the particular specification of the alphanumeric pager in use. The pagers typically deliver one to four lines of alphanumeric data. For the CSMS, the page includes the name of the sender, name of the affected host and the timestamp of the failure. The sender is the account name under which the *crontab* script is running. The golf course CSMS runs under the separate account 'monitoring' and is pinged from the Computer Science and System Technology Lab LANs. A bonafide failure of the high-value host will be that the administrator wearing the pager will receive two e-mail messages from the two monitoring hosts in short succession. Below is a sample notification.

```
monitoring@cs indy-gc.cs.nps.navy.mil is inactive
Thu Aug 8 14:45:50 PDT 1996
```

```
monitoring@stl indy-gc.cs.nps.navy.mil is inactive
Thu Aug 8 14:46:12 PDT 1996
```

monitoring@cs and monitoring@stl are both running the monitoring script.

When the host fails to respond to a *ping*, the *pinging* hosts send a e-mail messages to the address of the monitoring account that has the e-mail to alphanumeric paging operation.

## **F. AUTOMATIC ARCHIVAL OF MBONE**

During this thesis, the Mbone VCR application was in beta testing and was not configured for automatic launch on boot up. This is a necessary part of the CSMS.



Without it, the system is functional only if un-interrupted operation can be assured (i.e. it is vulnerable to power failure). The automatic archival and startup of the Mbone VCR is recommended for future work.

## **G. SUMMARY**

Network monitoring is an important part of network security. The CSMS shows that monitoring can be a powerful deterrent to physical threats to a network. The damage which occurs in the event of a breach of security can be greatly diminished by a well throughout security monitoring configuration.

The golf course CSMS proved that monitoring of a site can accomplished by using MBone VCR to record audio and video signals to a remote location. System installation does not require hardware investment other than an inexpensive video camera and card. The software is public-domain. Additional hard disk storage may be required. The system can function continuously and provide notification via email or e-mail to alphanumeric pager in the event of an alarm. A number of variables exist depending on the network configuration particularly mail server and mrouter configurations.



## **VII. EXPERIMENTAL RESULTS**

### **A. INTRODUCTION**

Networks are a significant part of the DoD. The Navy in particular is a unique networking environment because it is inherently self-supporting. There is no vendor technical support at sea. This forces the novice class of network user from existence. The Navy must have users who can employ basic monitoring techniques. The results of this paper show that the goal of monitoring self-sufficiency is a worthy and attainable goal.

Network monitoring is a difficult endeavor. This work has shown that monitoring is not difficult from the technical standpoint, but rather from an administrative standpoint. It requires persistence and determination. Computers do the same thing over and over. There is a reason behind every anomaly that presents itself. The cause of the anomaly must be determined not glossed over or ignored. Without an understanding of network monitoring and performance evaluation, network behavior is left to chance.

### **B. NPS**

NPS network monitoring design and implementation were successful. The automated status, the interactive tools and the notification procedures are all in place and available for all users. Network monitoring has become an integral part of the network infrastructure. A dedicated CGI server, `blackand.stl.nps.navy.mil`, continues to provide a

flexible and secure WWW CGI environment. Future Web application development will benefit from a well configured design environment and willing support staff.

### C. AUV 96

The IEEE AUV 96 Conference was held at the Monterey Hyatt Hotel June 3 - June 6. The IIRG set the goal of transmitting the entire conference worldwide via MBone. Accompanying this goal was the goal of monitoring the temporary network throughout its short existence. Both goals were achieved.

The Conference was held at the Hyatt hotel which is approximately one-half mile east of NPS. The two are separated by Highway 1 which is a four-lane freeway. It was not possible to have fiber connectivity from NPS to the Hyatt. To connect the Hyatt to NPS an AIRLAN wireless bridge was positioned atop the highest building at NPS and the roof of the Hyatt. As a precautionary measure, an auxiliary AIRLAN was configured as a backup.

The AUV temporary network included three Silicon Graphics workstations. The workstations were connected to the Computer Science LAN at NPS via the AIRLAN. Although physically separate, the AUV nodes functioned logically as part of the graphics subnet of the Computer Science LAN.

Each side of the AIRLAN had it's own IP address as did each host at the Hyatt. Even though all of these addresses were temporary, they were fully monitored for the entire time of their activity. The monitoring scripts developed for the System Technology Lab at NPS were ported to a Computer Science host and the host file reconfigured. The

hosts of interest were not just the temporary hosts but all hosts necessary to deliver IP and MBone to the off-campus router. The AUV conference had it's own monitoring page which proved extremely useful in the numerous trouble shooting efforts which occurred. This page continues to be in operation and is available at <http://www.stl.nps.navy.mil/~auv/monitoring/>

From the beginning, liaison with Computer Science department personnel was essential. Proper and complete monitoring occurred because network addressing and routing resolution were correct and known in advance. Without a doubt, success at AUV 96 depended not just on those directly responsible but on numerous personnel who performed a variety of functions. AUV 96 is further covered in [Erdogan, 96], [Tamer, 96] and [Tiddy, 96].

The physical security of the network assets was a serious consideration. 24 hour-a-day presence was required at the remote site. This monitoring provided a valuable backup to occasional hotel security checks. In the future, audio/video site monitoring can be accomplished via remote MBone archival as delineated in chapter 6.

#### **D. BAYNET**

BayNet IP monitoring has been successfully in-place at NPS and UCSC for several months. BayNet status as seen from UCSC has been continuously available at <http://www.cse.ucsc.edu/~edwardse/monitoring>. The goal of monitoring BayNet ATM connectivity was not achieved. ATM monitoring failed not because of a failure of the

monitoring suite itself, either in functionality or portability, but because of the downfall of BayNet ATM itself.

Moss Landing Marine Lab was configured in August, 1996. Porting the software took less than one hour. The software installation instructions are included as Appendix Q. This validates the ease of installation and use of public domain tools and software and HTTP. [McClean, 96]

Two applications ran somewhat successfully on BayNet: BayLink and UCSC distance learning. BayLink displayed real-time undersea video from MBARI research vessels to the Monterey Bay Aquarium and the San Jose Technical Museum of Innovation. Aquarium and museum personnel operated BayLink as a collateral duty, mostly without training. A genuine need for crystal-clear monitoring procedures was evident. The commitment to ATM waned as the end of the CalRen grant neared. And as it did, so did the desire to expend any more effort into BayLink ATM. The BayLink application enjoyed only moderate success, with users reporting frustration due to frequent system failures. Had sufficient monitoring tools been given to the users, an increase in the BayLink mission accomplishment might have resulted.

UCSC distance learning was partially successful. The two campuses, UCSC and UCSC Extension Santa Clara, are both located within the northern portion of BayNet. SVC packet delivery was accomplished through a single PacBell ATM switch. The rest of BayNet was not able to employ SVCs because of the lack of interoperability of the proprietary ATM switches. ATM standards are incomplete. This lack of standards created the unacceptable circumstance that BayNet cannot establish SVCs across multiple

switches. UCSC Extension has decided not to continue with the ATM-based distance learning due to the relative expense.

Another fatal flaw in BayNet was the lack of clearly defined goals and an accompanying timeline. BayNet involved participants from numerous institutions and organizations. A great deal of momentum was generated but without shared consensus on direction. As a result of this failure, NPS did not run a genuine application over BayNet. [Courtney, 96] The benefit of running an ATM-based application coupled with real-time monitoring and performance evaluation remains to be proven.

## **E. HUMAN INTERFACE**

Many of the problems that network users face are human problems. They must be answered with human solutions. This must be part of future solutions to monitoring problems. To ignore the importance of the human factor of the network equation can be a fatal flaw, as demonstrated in BayNet. Human interface requires three things:

### **1. Communication**

Communication is a word that is overused but underemployed. Minor changes to the BayNet configuration in some cases took over 6 months. The changes themselves rarely required more than 5 minutes of work. [Courtney, 96] Part of the delay was in talking to the wrong people. Communication is vital.

## **2. Authority**

The ownership of and authority of assets and the decision making power over them is upwardly mobile. The authority over assets belongs with those who use them the most.

In the construction of the STL monitoring package, several simple decisions had to be processed up the chain of command. Simple decisions such as the access to a group account or the creation of a simple account must be resident at the lowest levels. Each step up the decision-making chain costs time and energy. Decision making power belongs at the lowest possible level for effective network management.

## **3. Accountability**

Without a dedicated effort, effective monitoring will not happen. Rather than being a special use appliance for occasional utilization, monitoring needs to become part of the networking environment, both for users and managers. This requires that the effective communication and the authority to make decisions (as previously mentioned) exists. Effective monitoring requires involvement by more than just the system administrator. Our experience is that many users and administrators are glad to use these tools on a regular basis once they understand that they are available.

## **F. SUMMARY**

Monitoring is possible. Monitoring is important. Monitoring can happen without impact to network resources. Monitoring requires dedicated effort and a fundamental change to networking philosophy.



Network monitoring was successfully integrated into the NPS STL. The tools ported easily to AUV 96, the golf course lab and other NPS LANs. The monitoring applications installed easily at UCSC and MLML. They have been shown to be easy to use and access without impact on system security or performance. HTML monitoring home pages have proven to be the interface of choice, requiring little or no instruction on use. Testing and analysis of network status occurred at speeds a magnitude smaller than command line interaction.



## VIII. CONCLUSIONS AND RECOMMENDATIONS

### A. INTRODUCTION

Network monitoring is a vast frontier. The phenomenal growth of networking and the increased dependence on networks throughout business, military and academic pursuits require that they be reliable. To be reliable, they must be measured and analyzed as a normal course of operation. Currently, network reliability is taken as it comes (for better or worse).

### B. CONCLUSIONS

Effective monitoring is possible without large resource expenditure. There is no single answer to the complex monitoring problem but general guidelines suggest that effective monitoring is part active, part passive, part static and part interactive. It is essential to gather data on the status of network assets on a regular basis and garner the ability to analyze and display it. Users must have access to both historic and real-time reports on the host status, availability and topology. Smooth advances in internetworking in the future demand this user-aware approach. Monitoring is important not just for experts but for all network users. *ping*, *traceroute* and *nslookup* are tools most users can benefit from when given a user-friendly interface.

Packet filters are the only way to adequately and accurately test throughput. But like any other tool they are not fool-proof. All results must be verified. *tcpdump* is cryptic but it is the tool of choice for packet filtering. We expect other tools to be

available soon. Packet filters must be reserved for privileged users (due to privacy reasons) but they must be used. Other tools produce throughput information but the information can be misleading in numerous subtle ways.

HTTP is the protocol of choice for scripting applications. Reliance on HTTP makes applications extensible and portable. User access to HTTP and HTML, (i.e. web browsers), is an efficient and effective user interface. HTTP and HTML are standardized and benefit from an ongoing standards process. It is undeniable that command-line interaction is enough to discourage many users involvement with network monitoring. By using browsers to interface to the command line via well-constructed robust scripting, this major obstacle is overcome.

The one-stop network monitoring proposition is well-suited for most networks. Internetworking is not a blind march of technological advance, rather it is an ongoing synthesis of small technological achievements coupled with a lot of human interaction. Providing a centrally located monitoring location is akin to going to the mall: everything necessary in one place. Monitoring results, interactive testing, human points of contact, topology, changes to procedures, manual (man) pages, text versions of monitoring code, operating system specifics, hardware specifications, software versions, and even system formats presented in a layered selection process eliminate a great deal of the guesswork currently required to monitor a network. For all but expert local users continuing diagnostic troubleshooting, it will hopefully be all that is required.

The lack of network documentation at NPS is a severe problem. This recognized problem is totally unprofessional. Because of it, topologies must be re-invented or

reverse engineered, host information is scarce and many simple questions must be handled in person.

Commercial monitoring solutions are the wrong answer. They are expensive. Often they provide results which many networkers don't know how to interpret. They leave the monitoring responsibility on the shoulders of the select few who know how to use the system. They are brittle and failure prone. Proprietary commercial monitoring solutions are not extensible nor are they portable. There is no doubt that widespread solutions to complex network monitoring problems do not include proprietary commercial systems.

Monitoring must be an inherent part of the network, not an add-on or an afterthought. Integration and implementation must not place unnecessary burden on network administration. Public-domain software exists which accomplishes this goal and more is becoming available. Network managers must take responsibility for policing their own network. Drivers do not operate their vehicles (hopefully) without looking at the dash or (worse yet) not even knowing that there is a dashboard. Networkers must start building a dashboard that is sufficient to the task, and then use it.

ATM and IP do not operate well together. Lack of effective ATM standards compliance cripples interoperability. Inflexibility decreases the ability to rely on ATM from long-haul networking ventures. Lastly, the complexity of ATM causes uncertainty in ATM endeavors.

Network security is greatly enhanced through network monitoring. Physical monitoring such as the Continuous Site Monitoring System (CSMS) provides obvious

protection. Great security gains can be made by simply becoming familiar with the behavior of the network. Unexplainable occurrences will stand out and users will quickly notice unusual behavior. Subsequently less time will be squandered looking in the wrong direction when a security breach occurs.

### **C. RECOMMENDATIONS FOR FUTURE WORK**

To ensure reliability through timely fault notification and provide real-time network availability, network monitoring capability must be inherent to all networks regardless of size. The following recommendations for future work are suggested to enhance that pursuit.

#### **1. NPS**

##### ***a. campus-wide monitoring***

Develop and implement continuous monitoring of all NPS subnets. Provide a centralized location to house all monitoring related data, technical and administrative.

##### ***b. Documentation***

The NPS internetwork is in desperate need of documentation. The correct documentation of the campus subnet structure is vital to well organized and efficient networking ventures in the future.

## **2. BayNet**

The BayNet ATM network may well be doomed. BayNet however will continue to function in other regards. The monitoring capability can easily continue and undergo enhancements as desired. As in the case of the NPS campus network, documentation is important to future endeavors. Further automation can occur through the use of CGI scripts to pass current domain status to other network domains. The testing scripts themselves can be written to automatically update to the current version when required. Further development of BayNet monitoring will carry on in areas other than ATM such as Mbone, IPV6 and VRTP.

## **3. Remote Monitor Management Information Base**

Proponents of Simple Network Monitoring Protocol (SNMP) and Remote Monitor (RMON) Management Information Bases (MIBs) expect them to automate LAN diagnosis and disaster recovery. The great hope of RMON is that detailed performance analysis can become a normal part of network management. SNMP/RMON will provide statistical analysis, error tracing and an automatic alarm feature. This technology is crucial to the future of the Internetworking and is an exciting area of research to pursue.

## **4. Virtual Reality Modeling Protocol (VRTP)**

HTTP changed the face of networking. No one knows what the next leap will be. The Virtual Reality Modeling Language (VRML) and the associated protocol VRTP promise to transform the Internet in much the same way as did HTTP. [Brutzman, 96]

A sensible approach to the introduction of VRTP is to include a network management model in the protocol rather than reverse engineer it. Benefits lasting the entire lifespan of the protocol will be realized with the inclusive ability to resolve the optimum path, enable automatic version comparison and update, and determine when and where re-routing is required. Packet loss information, throughput, queuing statistics, garbage collection and a wide range of functionality that has traditionally been an afterthought can be assembled into the basic protocol.



## **APPENDIX A. DEFINITIONS**

### **ADDRESS RESOLUTION PROTOCOL (ARP) SERVER**

An ARP Server is a server like any other server except it has the additional duty of managing it's ARP cache. In order for a host to connect to another host, it must first determine the other host's NSAP address. The ARP procedure is used to resolve an IP address into an ATM address. [Fore, 95]

### **ASYNCHRONOUS TRANSFER MODE (ATM)**

Asynchronous Transfer Mode. International standard for cell relay in which multiple service types (such as voice, video, or data) are conveyed in fixed-length (53-byte) cells. Fixed-length cells allow cell processing to occur in hardware, thereby reducing transit delays. ATM is designed to take advantage of high-speed transmission media such as E3, SONET, and T3. [Cisco, 95]

### **BANDWIDTH**

Bandwidth is the difference between the limiting frequencies of a continuous frequency spectrum. [National Communications Systems, 86]

### **BAY AREA NETWORK (BAYNET)**

In early 1994 a number of grants were funded that had been collaboratively planned to enable several independent organizations and volunteer groups to design and implement a regional wide area network called Monterey Bay Area Network. BayNet is focused in the Education, Health Care, Community, Government and Commercial Business area. Projects are funded for two years, through October 1996, using PacBell data communications technologies. Technologies available include Integrated Services Digital Network (ISDN), Switched Digital Service-56, Switched Multimegabit Data Service (SMDS), Frame Relay and ATM. [Bigelow, 96]

### **CELL**

A cell is small fixed-size packet. [Partridge, 94]

## **COMMON GATEWAY INTERFACE (CGI)**

“CGI (Common Gateway Interface) is the interface between your Web site's HTTP (HyperText Transport Protocol) server and the other resources of your server's host computer.” [Weinman, 95]

## **INTERNET PROTOCOL (IP)**

IP is a network layer protocol in the TCP/IP stack offering a connectionless-routed internetwork service. IP provides features for addressing, type-of-service specification, fragmentation and reassembly, and security. [Cisco, 95]

## **INTERNETWORK**

An internetwork is a connection of multiple LANs. [Cisco, 95]

## **LATA**

Local Access and Transport Area. Geographic telephone dialing area serviced by a single local telephone company. Calls within LATAs are called "local calls," calls across LATA boundaries are "long distance" calls. There are over 100 LATAs in the United States. [Cisco, 95]

## **LOCAL AREA NETWORK (LAN)**

A LAN is a general purpose network that is confined to a small area such as a single building or a small cluster of buildings. LANs are generally owned and operated by a single organization. [Stallings, 91]

## **MAXIMUM TRANSMISSION UNIT (MTU)**

Maximum packet size (in bytes) that a particular interface can handle. “Classical IP” for ATM uses a default MTU of 9,180 bytes. MTU for Ethernet LAN Emulation is 1,500 bytes. The maximum MTU for 10 Mbps Ethernet and for 100 Mbps Fast Ethernet is 1,500 bytes. The maximum MTU for 4 Mbps Token Ring is 4,472 bytes and for 16 Mbps Token Ring is 17,800 bytes. The maximum MTU for 100 Mbps FDDI is 4,500 bytes. [Cisco, 95]

## **MULTICAST BACKBONE (MBone)**

Multicast functionality lies between unicast and broadcast. It is essentially a one-to-many or many-to-many addressing capability. The Multicast Backbone is a virtual network which overlays the Internet using a series of dedicated routers and hosts.

[Macedonia, Brutzman, 94]

## **NSAP ADDRESS**

The Network Service Access Point or NSAP address uniquely identifies ATM endpoints. The NSAP address consists of a 13 byte network-side prefix and a seven byte End System Identifier (ESI). The ESI is the unique IEEE MAC address of the interface.

[Alles, 95]

## **OSI REFERENCE MODEL**

Open System Interconnection reference model. Network architectural model developed by ISO and ITU-T. The model consists of seven layers, each of which specifies particular network functions such as addressing, flow control, error control, encapsulation, and reliable message transfer. The highest layer (the application layer) is closest to the user; the lowest layer (the physical layer) is closest to the media technology. The lower two layers are implemented in hardware and software, while the upper five layers are implemented only in software. The OSI reference model is used universally as a method for teaching and understanding network functionality. Similar in some respects to SNA. In practice, network protocols do not always map clearly to the partitioned layers of the OSI Reference Model. [Cisco, 95]

## **PACKET**

A packet is a specific amount of data with control information attached to it.

[Partridge, 94]

## **PERMANENT VIRTUAL CIRCUIT (PVCS)**

Virtual circuit that is permanently established. PVCs save bandwidth associated with circuit establishment and tear down in situations where certain virtual circuits must exist all the time. Called a permanent virtual connection in ATM terminology.

[Cisco, 95]

## **SWITCHED VIRTUAL CIRCUIT (SVCS)**

An SVCs is a circuit that is dynamically established on demand and is torn down when transmission is complete. SVCs are used in situations where data transmission is sporadic. [Cisco, 95]

## **TRANSMISSION MEDIA**

Transmission medium is the physical path between stations on a network. Examples include copper, air, fiber optic cable and microwave. [National Communications Systems, 86]

## APPENDIX B. *ping* MAN PAGE (UNIX)

PING(1M)

PING(1M)

### NAME

ping - send ICMP ECHO\_REQUEST packets to network hosts

### SYNOPSIS

```
/usr/etc/ping [-dfnqrVRL] [-c count] [-s size] [-l preload]
               [-i interval] [-p pattern] [-T ttl] [-I addr] host
```

### DESCRIPTION

Ping is a tool for network testing, measurement and management. It utilizes the ICMP protocol's ECHO\_REQUEST datagram to elicit an ICMP ECHO\_RESPONSE from a host or gateway. ECHO\_REQUEST datagrams ('pings') have an IP and ICMP header, followed by an 8-byte timestamp, and then an arbitrary number of 'pad' bytes used to fill out the packet.

The host can be the name of a host or its Internet address. The options are:

- c count  
Stop after sending (and receiving) count ECHO\_RESPONSE packets.
- d Set the SO\_DEBUG option on the socket being used.
- f Flood ping. Outputs packets as fast as they come back or one hundred times per second, whichever is more. (The repetition rate can be adjusted with the -i option.) For every ECHO\_REQUEST sent a period '.' is printed, while for every ECHO\_REPLY received a backspace is printed. This provides a rapid display of how many packets are being dropped. This can be extremely stressful on a network and should be used with caution.
- i interval  
Wait interval seconds between sending each packet. The default is to wait for one second between each packet, except when the -f option is used when the default is 0.01 second.
- l preload  
Send preload packets as fast as possible before falling into the normal mode of behavior.
- n Numeric output only. No attempt will be made to lookup symbolic names for host addresses. Useful if your name server is flaky or for hosts not in the database.
- p pattern  
You may specify up to 16 'pad' bytes to fill out the packet you send. This is useful for diagnosing data-dependent problems in a network. For example, '-p ff' will cause the sent packet to be filled with all ones.
- q Quiet output. Nothing is displayed except the summary line on termination.

Page 1

## PING(1M)

## PING(1M)

- r Bypass the normal routing tables and send directly to a host on an attached network. If the host is not on a directly-attached network, an error is returned. This option can be used to ping a local host through an interface that has no route through it (e.g., after the interface was dropped by routed(1M)).
- s size  
Send datagrams containing size bytes of data. The default is 56, which translates into 64 ICMP data bytes when combined with the 8 bytes of ICMP header data. The maximum allowed value is 65468 bytes.
- v Verbose output. ICMP packets other than ECHO RESPONSE that are received are listed.
- I interface  
Send multicast datagrams on the network interface specified by the interface's hostname or IP address.
- L When sending to a multicast destination address, don't loop the datagram back to ourselves.
- R Record Route. Includes the RECORD\_ROUTE option in the ECHO\_REQUEST packet and displays the route buffer on returned packets. Note that the IP header is only large enough for six such routes. Many hosts ignore or discard this option.
- T ttl  
Changes the default time-to-live for datagrams sent to a multicast address.

Ping should be used primarily for manual fault isolation. Because of the load it can impose on the network, it is unwise to use ping during normal operations or from automated scripts. When using ping for fault isolation, it should first be run on the local host, to verify that the local network interface is up and running. Then, hosts and gateways further and further away should be ``pinged''.

Ping continually sends one datagram per second, and prints one line of output for every ECHO\_RESPONSE returned. On a trusted system with IP Security Options enabled, if the network idiom is not MONO, ping also prints a second line containing the hexadecimal representation of the IP security option in the ECHO\_RESPONSE. If the -c count option is given, only that number of requests is sent. No output is produced if there is no response. Round-trip times and packet loss statistics are computed. If duplicate packets are received, they are not included in the packet loss calculation, although the round trip time of these packets is used in calculating the minimum/average/maximum round-trip time numbers. When the specified number of packets have been sent (and received) or if the program is terminated with an interrupt (SIGINT), a brief summary is displayed. When not using the -f (flood) option, the first interrupt, usually generated by control-C or DEL, causes ping to wait for its

PING(1M)

PING(1M)

outstanding requests to return. It will wait no longer than the longest round trip time encountered by previous, successful pings. The second interrupt stops ping immediately.

#### DETAILS

An IP header without options is 20 bytes. An ICMP ECHO\_REQUEST packet contains an additional 8 bytes worth of ICMP header followed by an arbitrary amount of data. When a packetsize is given, this indicates the size of this extra piece of data (the default is 56). Thus the amount of data received inside of an IP packet of type ICMP ECHO\_REPLY will always be 8 bytes more than the requested data space (the ICMP header).

If the data space is at least eight bytes large, ping uses the first eight bytes of this space to include a timestamp which it uses in the computation of round trip times. If less than eight bytes of pad are specified, no round trip times are given.

#### DUPLICATE AND DAMAGED PACKETS

Ping will report duplicate and damaged packets. Duplicate packets should never occur, and seem to be caused by inappropriate link-level retransmissions. Duplicates may occur in many situations and are rarely (if ever) a good sign, although the presence of low levels of duplicates may not always be cause for alarm.

Damaged packets are obviously serious cause for alarm and often indicate broken hardware somewhere in the ping packet's path (in the network or in the hosts).

#### TRYING DIFFERENT DATA PATTERNS

The (inter)network layer should never treat packets differently depending on the data contained in the data portion. Unfortunately, data-dependent problems have been known to sneak into networks and remain undetected for long periods of time. In many cases the particular pattern that will have problems is something that doesn't have sufficient "transitions", such as all ones or all zeros, or a pattern right at the edge, such as almost all zeros. It isn't necessarily enough to specify a data pattern of all zeros (for example) on the command line because the pattern that is of interest is at the data link level, and the relationship between what you type and what the controllers transmit can be complicated.

This means that if you have a data-dependent problem you will probably have to do a lot of testing to find it. If you are lucky, you may manage to find a file that either can't be sent across your network or that takes much longer to transfer than other similar length files. You can then examine this file for repeated patterns that you can test using the -p option of ping.

#### TTL DETAILS

The TTL value of an IP packet represents the maximum number of IP routers that the packet can go through before being thrown away. In current practice you can expect each router in the Internet to decrement the TTL field by exactly one.

Page 3

## PING(1M)

## PING(1M)

The TCP/IP specification says that the TTL field for TCP packets should be set to 60, but many systems use smaller values (IRIX and 4.3BSD use 30, 4.2BSD used 15).

The maximum possible value of this field is 255, and most Unix systems set the TTL field of ICMP ECHO\_REQUEST packets to 255. This is why you will find you can ``ping'' some hosts, but not reach them with telnet or ftp.

In normal operation ping prints the ttl value from the packet it receives. When a remote system receives a ping packet, it can do one of three things with the TTL field in its response:

- o Not change it; this is what Berkeley Unix systems did before the 4.3BSD-tahoe release. In this case the TTL value in the received packet will be 255 minus the number of routers in the round-trip path.
- o Set it to 255; this is what IRIX and current Berkeley Unix systems do. In this case the TTL value in the received packet will be 255 minus the number of routers in the path from the remote system to the pinging host.
- o Set it to some other value. Some machines use the same value for ICMP packets that they use for TCP packets, for example either 30 or 60. Others may use completely wild values.

## BUGS

Many Hosts and Gateways ignore the RECORD\_ROUTE option.

The maximum IP header length is too small for options like RECORD\_ROUTE to be completely useful. There's not much that can be done about this, however.

Flood pinging is not recommended in general, and flood pinging the broadcast address should only be done under very controlled conditions.

The record-route option does not work with hosts using network code derived from 4.3BSD.

## SEE ALSO

netstat(1), ifconfig(1M), routed(1M)



## APPENDIX C. *ftp* MAN PAGE (UNIX)

FTP(1C)

FTP(1C)

### NAME

*ftp* - Internet file transfer program

### SYNOPSIS

*ftp* [ -v ] [ -d ] [ -i ] [ -n ] [ -g ] [ host ]

### DESCRIPTION

*Ftp* is the user interface to the Internet standard File Transfer Protocol. The program allows a user to transfer files to and from a remote network site.

The client host with which *ftp* is to communicate may be specified on the command line. If this is done, *ftp* will immediately attempt to establish a connection to an FTP server on that host; otherwise, *ftp* will enter its command interpreter and await instructions from the user. When *ftp* is awaiting commands from the user the prompt "*ftp*>" is provided to the user. The following commands are recognized by *ftp*:

! [ command [ args ] ]

Invoke an interactive shell on the local machine. If there are arguments, the first is taken to be a command to execute directly, with the rest of the arguments as its arguments.

\$ macro-name [ args ]

Execute the macro macro-name that was defined with the *macdef* command. Arguments are passed to the macro unglobbed.

account [ passwd ]

Supply a supplemental password required by a remote system for access to resources once a login has been successfully completed. If no argument is included, the user will be prompted for an account password in a non-echoing input mode.

append local-file [ remote-file ]

Append a local file to a file on the remote machine. If *remote-file* is left unspecified, the local file name is used in naming the remote file after being altered by any *ntrans* or *nmap* setting. File transfer uses the current settings for type, format, mode, and structure.

ascii

Set the file transfer type to network ASCII. This is the default type if *ftp* cannot determine the type of operating system running on the remote machine or the remote operating system is not UNIX.

bell Arrange that a bell be sounded after each file transfer command is completed.

binary

Set the file transfer type to support binary image transfer. This is the default type if *ftp* can determine that the remote machine is running UNIX.

Page 1

- bye** Terminate the FTP session with the remote server and exit ftp. An end of file will also terminate the session and exit.
- case** Toggle remote computer file name case mapping during mget commands. When case is on (default is off), remote computer file names with all letters in upper case are written in the local directory with the letters mapped to lower case.
- cd remote-directory**  
Change the working directory on the remote machine to remote-directory.
- cdup** Change the remote machine working directory to the parent of the current remote machine working directory.
- chmod mode file-name**  
Change the permission modes for the file file-name on the remote system to mode.
- close**  
Terminate the FTP session with the remote server, and return to the command interpreter. Any defined macros are erased.
- cr** Toggle carriage return stripping during ascii type file retrieval. Records are denoted by a carriage return/linefeed sequence during ascii type file transfer. When cr is on (the default), carriage returns are stripped from this sequence to conform with the UNIX single linefeed record delimiter. Records on non-UNIX remote systems may contain single linefeeds; when an ascii type transfer is made, these linefeeds may be distinguished from a record delimiter only when cr is off.
- delete remote-file**  
Delete the file remote-file on the remote machine.
- debug [ debug-value ]**  
Toggle debugging mode. If an optional debug-value is specified it is used to set the debugging level. When debugging is on, ftp prints each command sent to the remote machine, preceded by the string "--->".
- dir [ remote-directory ] [ local-file ]**  
Print a listing of the directory contents in the directory, remote-directory, and, optionally, placing the output in local-file. If interactive prompting is on, ftp will prompt the user to verify that the last argument is indeed the target local file for receiving dir output. If no directory is specified, the current working directory on the remote machine is used. If no local file is specified, or local-file is -, output comes to the terminal.

**disconnect**  
A synonym for close.

**form format**  
Set the file transfer form to format. The default format is "file".

**get remote-file [ local-file ]**  
Retrieve the remote-file and store it on the local machine. If the local file name is not specified, it is given the same name it has on the remote machine, subject to alteration by the current case, ntrans, and nmap settings. The current settings for type, form, mode, and structure are used while transferring the file.

**glob** Toggle filename expansion for mdelete, mget and mput. If globbing is turned off with glob, the file name arguments are taken literally and not expanded. Globbing for mput is done as in csh(1). For mdelete and mget, each remote file name is expanded separately on the remote machine and the lists are not merged. Expansion of a directory name is likely to be different from expansion of the name of an ordinary file: the exact result depends on the foreign operating system and ftp server, and can be previewed by doing `'mll remote-files -'`. Note: mget and mput are not meant to transfer entire directory subtrees of files. That can be done by transferring a tar(1) archive of the subtree (in binary mode).

**hash** Toggle hash-sign ('`#`') printing for each data block transferred. The size of a data block is 1024 bytes.

**help [ command ]**  
Print an informative message about the meaning of command. If no argument is given, ftp prints a list of the known commands.

**idle [ seconds ]**  
Set the inactivity timer on the remote server to seconds seconds. If seconds is omitted, the current inactivity timer is printed.

**lcd [ directory ]**  
Change the working directory on the local machine. If no directory is specified, the user's home directory is used.

**ls [ remote-directory ] [ local-file ]**  
Print a listing of the contents of a directory on the remote machine. The listing includes any system-dependent information that the server chooses to include; for example, most UNIX systems will produce output from the command `"ls -lA"`. (See also nlist.) If remote-directory is left unspecified, the current working directory is used. If interactive prompting is on, ftp will prompt the user to verify that the last argument is indeed the target local file for receiving ls output. If no local file is specified, or if local-file is -, the output is sent to the terminal.

**macdef macro-name**  
 Define a macro. Subsequent lines are stored as the macro macro-name; a null line (consecutive newline characters in a file or carriage returns from the terminal) terminates macro input mode. There is a limit of 16 macros and 4096 total characters in all defined macros. Macros remain defined until a close command is executed. The macro processor interprets '\$' and '\' as special characters. A '\$' followed by a number (or numbers) is replaced by the corresponding argument on the macro invocation command line. A '\$' followed by an 'i' signals that macro processor that the executing macro is to be looped. On the first pass '\$i' is replaced by the first argument on the macro invocation command line, on the second pass it is replaced by the second argument, and so on. A '\' followed by any character is replaced by that character. Use the '\' to prevent special treatment of the '\$'.

**mdelete [ remote-files ]**  
 Delete the remote-files on the remote machine.

**mdir remote-files local-file**  
 Like dir, except multiple remote files may be specified. If interactive prompting is on, ftp will prompt the user to verify that the last argument is indeed the target local file for receiving mdir output.

**mget remote-files**  
 Expand the remote-files on the remote machine and do a get for each file name thus produced. See glob for details on the filename expansion. Resulting file names will then be processed according to case, ntrans, and nmap settings. Files are transferred into the local working directory, which can be changed with 'lcd directory'; new local directories can be created with '! mkdir directory'.

**mkdir directory-name**  
 Make a directory on the remote machine.

**mls remote-files local-file**  
 Like nlist, except multiple remote files may be specified, and the local-file must be specified. If interactive prompting is on, ftp will prompt the user to verify that the last argument is indeed the target local file for receiving mls output.

**mode [ mode-name ]**  
 Set the file transfer mode to mode-name. The default mode is "stream" mode.

**modtime file-name**  
 Show the last modification time of the file on the remote machine.

**mput local-files**  
 Expand wild cards in the list of local files given as arguments and do a put for each file in the resulting list. See glob for details

of filename expansion. Resulting file names will then be processed according to ntrans and nmap settings.

#### newer file-name

Get the file only if the modification time of the remote file is more recent than the file on the current system. If the file does not exist on the current system, the remote file is considered newer. Otherwise, this command is identical to get.

#### nlist [ remote-directory ] [ local-file ]

Print a list of the files of a directory on the remote machine. If remote-directory is left unspecified, the current working directory is used. If interactive prompting is on, ftp will prompt the user to verify that the last argument is indeed the target local file for receiving nlist output. If no local file is specified, or if local-file is -, the output is sent to the terminal.

#### nmap [ inpattern outpattern ]

Set or unset the filename mapping mechanism. If no arguments are specified, the filename mapping mechanism is unset. If arguments are specified, remote filenames are mapped during mput commands and put commands issued without a specified remote target filename. If arguments are specified, local filenames are mapped during mget commands and get commands issued without a specified local target filename. This command is useful when connecting to a non-UNIX remote computer with different file naming conventions or practices. The mapping follows the pattern set by inpattern and outpattern. Inpattern is a template for incoming filenames (which may have already been processed according to the ntrans and case settings). Variable templating is accomplished by including the sequences '\$1', '\$2', ..., '\$9' in inpattern. Use '\' to prevent this special treatment of the '\$' character. All other characters are treated literally, and are used to determine the nmap inpattern variable values. For example, given inpattern \$1.\$2 and the remote file name "mydata.data", \$1 would have the value "mydata", and \$2 would have the value "data". The outpattern determines the resulting mapped filename. The sequences '\$1', '\$2', ..., '\$9' are replaced by any value resulting from the inpattern template. The sequence '\$0' is replaced by the original filename. Additionally, the sequence '[seq1,seq2]' is replaced by seq1 if seq1 is not a null string; otherwise it is replaced by seq2. For example, the command "nmap \$1.\$2.\$3 [\$1,\$2].[\$2,file]" would yield the output filename "myfile.data" for input filenames "myfile.data" and "myfile.data.old", "myfile.file" for the input filename "myfile", and "myfile.myfile" for the input filename ".myfile". Spaces may be included in outpattern, as in the example: nmap \$1 |sed "s/ \*\$//" > \$1. Use the '\' character to prevent special treatment of the '\$', '[', ']', and ',' characters.

#### ntrans [ inchars [ outchars ] ]

Set or unset the filename character translation mechanism. If no arguments are specified, the filename character translation

mechanism is unset. If arguments are specified, characters in remote filenames are translated during mput commands and put commands issued without a specified remote target filename. If arguments are specified, characters in local filenames are translated during mget commands and get commands issued without a specified local target filename. This command is useful when connecting to a non-UNIX remote computer with different file naming conventions or practices. Characters in a filename matching a character in inchars are replaced with the corresponding character in outchars. If the character's position in inchars is longer than the length of outchars, the character is deleted from the file name.

**open** host [ port ]

Establish a connection to the specified host FTP server. An optional port number may be supplied, in which case, ftp will attempt to contact an FTP server at that port. If the auto-login option is on (default), ftp will also attempt to automatically log the user in to the FTP server (see below).

**prompt**

Toggle interactive prompting. Interactive prompting occurs during multiple file transfers to allow the user to selectively retrieve or store files. If prompting is turned off (default is on), any mget or mput will transfer all files, and any mdelete will delete all files.

**proxy** ftp-command

Execute an ftp command on a secondary control connection. This command allows simultaneous connection to two remote ftp servers for transferring files between the two servers. The first proxy command should be an open, to establish the secondary control connection. Enter the command "proxy ?" to see other ftp commands executable on the secondary connection. The following commands behave differently when prefaced by proxy: open will not define new macros during the auto-login process, close will not erase existing macro definitions, get and mget transfer files from the host on the primary control connection to the host on the secondary control connection, and put, mput, and append transfer files from the host on the secondary control connection to the host on the primary control connection. Third party file transfers depend upon support of the ftp protocol PASV command by the server on the secondary control connection.

**put** local-file [ remote-file ]

Store a local file on the remote machine. If remote-file is left unspecified, the local file name is used after processing according to any ntrans or nmap settings in naming the remote file. File transfer uses the current settings for type, format, mode, and structure.

**pwd** Print the name of the current working directory on the remote machine.

quit A synonym for bye.

quote arg1 arg2 ...  
The arguments specified are sent, verbatim, to the remote FTP server.

recv remote-file [ local-file ]  
A synonym for get.

reget remote-file [ local-file ]  
Reget acts like get, except that if local-file exists and is smaller than remote-file, local-file is presumed to be a partially transferred copy of remote-file and the transfer is continued from the apparent point of failure. This command is useful when transferring very large files over networks that are prone to dropping connections.

remotehelp [ command-name ]  
Request help from the remote FTP server. If a command-name is specified it is supplied to the server as well.

remotestatus [ file-name ]  
With no arguments, show status of remote machine. If file-name is specified, show status of file-name on remote machine.

rename [ from ] [ to ]  
Rename the file from on the remote machine, to the file to.

reset  
Clear reply queue. This command re-synchronizes command/reply sequencing with the remote ftp server. Resynchronization may be necessary following a violation of the ftp protocol by the remote server.

restart marker  
Restart the immediately following get or put at the indicated marker. On UNIX systems, marker is usually a byte offset into the file.

rmdir directory-name  
Delete a directory on the remote machine.

runique  
Toggle storing of files on the local system with unique filenames. If a file already exists with a name equal to the target local filename for a get or mget command, a ".1" is appended to the name. If the resulting name matches another existing file, a ".2" is appended to the original name. If this process continues up to ".99", an error message is printed, and the transfer does not take place. The generated unique filename will be reported. Note that runique will not affect local files generated from a shell command (see below). The default value is off.

`send local-file [ remote-file ]`  
 A synonym for `put`.

`sendport`  
 Toggle the use of `PORT` commands. By default, `ftp` will attempt to use a `PORT` command when establishing a connection for each data transfer. The use of `PORT` commands can prevent delays when performing multiple file transfers. If the `PORT` command fails, `ftp` will use the default data port. When the use of `PORT` commands is disabled, no attempt will be made to use `PORT` commands for each data transfer. This is useful for certain FTP implementations which do ignore `PORT` commands but, incorrectly, indicate they've been accepted.

`site arg1 arg2 ...`  
 The arguments specified are sent, verbatim, to the remote FTP server as a `SITE` command.

`size file-name`  
 Return size of `file-name` on remote machine.

`status`  
 Show the current status of `ftp`.

`struct [ struct-name ]`  
 Set the file transfer structure to `struct-name`. By default "stream" structure is used.

`sunique`  
 Toggle storing of files on remote machine under unique file names. Remote `ftp` server must support `ftp` protocol `STOU` command for successful completion. The remote server will report unique name. Default value is off.

`system`  
 Show the type of operating system running on the remote machine.

`tenex`  
 Set the file transfer type to that needed to talk to `TENEX` machines.

`trace`  
 Toggle packet tracing.

`type [ type-name ]`  
 Set the file transfer type to `type-name`. If no type is specified, the current type is printed. The default type is network ASCII.

`umask [ newmask ]`  
 Set the default umask on the remote server to `newmask`. If `newmask` is omitted, the current umask is printed.



user user-name [ password ] [ account ]

Identify yourself to the remote FTP server. If the password is not specified and the server requires it, ftp will prompt the user for it (after disabling local echo). If an account field is not specified, and the FTP server requires it, the user will be prompted for it. If an account field is specified, an account command will be relayed to the remote server after the login sequence is completed if the remote server did not require it for logging in. Unless ftp is invoked with "auto-login" disabled, this process is done automatically on initial connection to the FTP server.

verbose

Toggle verbose mode. In verbose mode, all responses from the FTP server are displayed to the user. In addition, if verbose is on, when a file transfer completes, statistics regarding the efficiency of the transfer are reported. By default, verbose is on.

? [ command ]

A synonym for help.

Command arguments which have embedded spaces may be quoted with quote (") marks.

#### ABORTING A FILE TRANSFER

To abort a file transfer, use the terminal interrupt key (usually Ctrl-C). Sending transfers will be immediately halted. Receiving transfers will be halted by sending a ftp protocol ABOR command to the remote server, and discarding any further data received. The speed at which this is accomplished depends upon the remote server's support for ABOR processing. If the remote server does not support the ABOR command, an "ftp>" prompt will not appear until the remote server has completed sending the requested file.

The terminal interrupt key sequence will be ignored when ftp has completed any local processing and is awaiting a reply from the remote server. A long delay in this mode may result from the ABOR processing described above, or from unexpected behavior by the remote server, including violations of the ftp protocol. If the delay results from unexpected remote server behavior, the local ftp program must be killed by hand.

#### FILE NAMING CONVENTIONS

Files specified as arguments to ftp commands are processed according to the following rules.

- 1) If the file name "-" is specified, the stdin (for reading) or stdout (for writing) is used.
- 2) If the first character of the file name is "|", the remainder of the argument is interpreted as a shell command. Ftp then forks a shell, using popen(3) with the argument supplied, and reads (writes) from the stdout (stdin). If the shell command includes spaces, the

Page 9

argument must be quoted; e.g., "`| ls -lt`". A particularly useful example of this mechanism is: "`dir |more`".

- 3) Failing the above checks, if `globbing` is enabled, local file names are expanded according to the rules used in the `cs(1)`; c.f. the `glob` command. If the `ftp` command expects a single local file (e.g., `put`), only the first filename generated by the `globbing` operation is used.
- 4) For `mget` commands and `get` commands with unspecified local file names, the local filename is the remote filename, which may be altered by a `case`, `ntrans`, or `nmap` setting. The resulting filename may then be altered if `runique` is on.
- 5) For `mput` commands and `put` commands with unspecified remote file names, the remote filename is the local filename, which may be altered by a `ntrans` or `nmap` setting. The resulting filename may then be altered by the remote server if `sunique` is on.

#### FILE TRANSFER PARAMETERS

The FTP specification specifies many parameters which may affect a file transfer. The type may be one of `"ascii"`, `"image"` (binary), `"ebcdic"`, and `"local byte size"` (for PDP-10's and PDP-20's mostly). `Ftp` supports the `ascii` and `image` types of file transfer, plus local byte size 8 for `tenex` mode transfers.

`Ftp` supports only the default values for the remaining file transfer parameters: `mode`, `form`, and `struct`.

#### OPTIONS

Options may be specified at the shell command line. Several options can be enabled or disabled with `ftp` commands.

The `-v` (verbose on) option forces `ftp` to show all responses from the remote server, as well as report on data transfer statistics.

The `-n` option restrains `ftp` from attempting "auto-login" upon initial connection. If auto-login is enabled, `ftp` will check the `.netrc` file (see below) in the user's home directory for an entry describing an account on the remote machine. If no entry exists, `ftp` will prompt for the remote machine login name (default is the user identity on the local machine), and, if necessary, prompt for a password and an account with which to login.

The `-i` option turns off interactive prompting during multiple file transfers.

The `-d` option enables debugging.

The `-g` option disables file name globbing.

## THE .netrc FILE

The .netrc file contains login and initialization information used by the auto-login process. It resides in the user's home directory. The following tokens are recognized; they may be separated by spaces, tabs, or new-lines:

## machine name

Identify a remote machine name. The auto-login process searches the .netrc file for a machine token that matches the remote machine specified on the ftp command line or as an open command argument. Once a match is made, the subsequent .netrc tokens are processed, stopping when the end of file is reached or another machine or a default token is encountered.

## default

This is the same as machine name except that default matches any name. There can be only one default token, and it must be after all machine tokens. This is normally used as:

```
default login anonymous password user@site
```

thereby giving the user automatic anonymous ftp login to machines not specified in .netrc. This can be overridden by using the -n flag to disable auto-login.

## login name

Identify a user on the remote machine. If this token is present, the auto-login process will initiate a login using the specified name.

## password string

Supply a password. If this token is present, the auto-login process will supply the specified string if the remote server requires a password as part of the login process. Note that if this token is present in the .netrc file for any user other than anonymous, ftp will abort the auto-login process if the .netrc is accessible by anyone besides the user (see below for the proper protection mode.)

## account string

Supply an additional account password. If this token is present, the auto-login process will supply the specified string if the remote server requires an additional account password, or the auto-login process will initiate an ACCT command if it does not. Note that if this token is present in the .netrc file, ftp will abort the auto-login process if the .netrc is accessible by anyone besides the user (see below for the proper protection mode.)

## macdef name

Define a macro. This token functions like the ftp macdef command functions. A macro is defined with the specified name; its contents begin with the next .netrc line and continue until a null line (consecutive new-line characters) is encountered. If a macro named init is defined, it is automatically executed as the last step in the auto-login process.

The error message

Error: .netrc file is readable by others.

means the file is ignored by ftp because the file's password and/or account information is unprotected. Use

```
chmod go-rwx .netrc
to protect the file.
```

SEE ALSO  
ftpd(1M)

#### BUGS

Correct execution of many commands depends upon proper behavior by the remote server.

An error in the treatment of carriage returns in the 4.2BSD UNIX ascii-mode transfer code has been corrected. This correction may result in incorrect transfers of binary files to and from 4.2BSD servers using the ascii type. Avoid this problem by using the binary image type.

## APPENDIX D. *traceroute* MAN PAGE (UNIX)

TRACEROUTE(1M)

TRACEROUTE(1M)

### NAME

*traceroute* - print the route packets take to a network host

### SYNOPSIS

```
/usr/etc/traceroute [ -g addr ] [ -l ] [ -m max_ttl ] [ -n ] [ -p port ]  
[ -q nqueries ] [ -r ] [ -s src_addr ] [ -t tos ]  
[ -w waittime ] host [ datalen ]
```

### DESCRIPTION

The Internet is a large and complex aggregation of network hardware, connected by gateways. Tracking the route your packets follow (or finding the miscreant gateway that's discarding your packets) can be difficult. *traceroute* utilizes the IP protocol ``time-to-live'' (TTL) field and attempts to elicit an ICMP TIME\_EXCEEDED response from each gateway along the path to some host.

The only mandatory parameter is the destination host name or IP address. The default probe datagram length is 40 bytes, but this may be increased by specifying the additional length (in bytes) after the destination host name.

The options are:

- g Enable the IP LSRR (Loose Source Record Route) option in addition to the TTL tests. This is useful for asking how somebody else, at *addr*, (either an IP address or a hostname) reaches a particular *tgt*.
- l Print the value of the TTL field in each received packet (this can be used to help detect asymmetric routing).
- m Set the maximum time-to-live (maximum number of hops) used in outgoing probe packets. The default is 30 hops.
- n Print hop addresses numerically rather than symbolically and numerically (saves a nameserver address-to-name lookup for each gateway found on the path).
- p Set the base UDP port number used in probes (default is 33434). *traceroute* hopes that nothing is listening on UDP ports *base* to *base+n hops-1* at the destination host (so an ICMP PORT\_UNREACHABLE message will be returned to terminate the route tracing). If something is listening on a port in the default range, this option can be used to pick an unused port range.
- q Set the number of probe packets to send. The default is 3 packets.
- r Bypass the normal routing tables and send directly to a host on an attached network. If the host is not on a directly attached network, an error is returned. This option can be used to ping a local host through an interface that has no route through it (for example, after the interface was dropped by *routed(1M)*).

Page 1

# TRACEROUTE(1M)

# TRACEROUTE(1M)

- s Use the following IP address (which must be given as a number, not a hostname) as the source address in outgoing probe packets. On hosts with more than one IP address, this option can be used to force the source address to be something other than the IP address of the interface the probe packet is sent on. If the IP address is not one of this machine's interface addresses, an error is returned and nothing is sent.
- t Set the type-of-service (TOS) in probe packets to the following value (default zero). The value must be a decimal integer in the range 0 to 255. This option can be used to see if different types-of-service result in different paths. Not all values of TOS are legal or meaningful: see the IP RFC for definitions. Useful values are probably -t 16 (low delay) and -t 8 (high throughput).
- v Verbose output. Received ICMP packets other than TIME\_EXCEEDED and PORT\_UNREACHABLEs are listed.
- w Set the time (in seconds) to wait for a response to a probe (default is 3 seconds).

This program attempts to trace the route an IP packet would follow to some Internet host by launching UDP probe packets with a small TTL then, listening for an ICMP TIME\_EXCEEDED reply from a gateway. The probes begin with a TTL of one and increase by one until an ICMP PORT\_UNREACHABLE message is received, which means we got to ``host'' or hit the maximum (which defaults to 30 hops but can be changed with the -m flag). Three probes (changed with -q flag) are sent at each TTL setting and a line is printed showing the TTL, address of the gateway and round trip time of each probe. If the probe answers come from different gateways, the address of each responding system will be printed. If there is no response within a 3-second timeout interval (changed with the -w flag), a ``\*'' is printed for that probe.

So that the destination host will not process the UDP probe packets, the destination port is set to an unlikely value. If someone on the destination is using that value, it can be changed with the -p flag.

A sample use and output might be:

```
% traceroute nis.nsf.net.
traceroute to nis.nsf.net (35.1.1.48), 30 hops max, 56 byte packet
 1 helios.ee.lbl.gov (128.3.112.1)  19 ms  19 ms  0 ms
 2 lilac-dmc.Berkeley.EDU (128.32.216.1)  39 ms  39 ms  19 ms
 3 lilac-dmc.Berkeley.EDU (128.32.216.1)  39 ms  39 ms  19 ms
 4 ccngw-ner-cc.Berkeley.EDU (128.32.136.23)  39 ms  40 ms  39 ms
 5 ccn-nerif22.Berkeley.EDU (128.32.168.22)  39 ms  39 ms  39 ms
 6 128.32.197.4 (128.32.197.4)  40 ms  59 ms  59 ms
 7 131.119.2.5 (131.119.2.5)  59 ms  59 ms  59 ms
 8 129.140.70.13 (129.140.70.13)  99 ms  99 ms  80 ms
 9 129.140.71.6 (129.140.71.6)  139 ms  239 ms  319 ms
10 129.140.81.7 (129.140.81.7)  220 ms  199 ms  199 ms
```

TRACEROUTE(1M)

TRACEROUTE(1M)

```
11 nic.merit.edu (35.1.1.48) 239 ms 239 ms 239 ms
```

Notice that lines 2 and 3 are the same because of a buggy kernel on the second hop system - lbl-csam.arpa - that forwards packets with a zero TTL (a bug in the distributed version of 4.3BSD). You have to guess what path the packets are taking cross-country since the NSFNet (129.140) doesn't supply address-to-name translations for its NSSes.

A more interesting example is:

```
% traceroute allspice.lcs.mit.edu.
traceroute to allspice.lcs.mit.edu (18.26.0.115), 30 hops max
 1 helios.ee.lbl.gov (128.3.112.1) 0 ms 0 ms 0 ms
 2 lilac-dmc.Berkeley.EDU (128.32.216.1) 19 ms 19 ms 19 ms
 3 lilac-dmc.Berkeley.EDU (128.32.216.1) 39 ms 19 ms 19 ms
 4 ccngw-ner-cc.Berkeley.EDU (128.32.136.23) 19 ms 39 ms 39 ms
 5 ccn-nerif22.Berkeley.EDU (128.32.168.22) 20 ms 39 ms 39 ms
 6 128.32.197.4 (128.32.197.4) 59 ms 119 ms 39 ms
 7 131.119.2.5 (131.119.2.5) 59 ms 59 ms 39 ms
 8 129.140.70.13 (129.140.70.13) 80 ms 79 ms 99 ms
 9 129.140.71.6 (129.140.71.6) 139 ms 139 ms 159 ms
10 129.140.81.7 (129.140.81.7) 199 ms 180 ms 300 ms
11 129.140.72.17 (129.140.72.17) 300 ms 239 ms 239 ms
12 * * *
13 128.121.54.72 (128.121.54.72) 259 ms 499 ms 279 ms
14 * * *
15 * * *
16 * * *
17 * * *
```

18 ALLSPICE.LCS.MIT.EDU (18.26.0.115) 339 ms 279 ms 279 ms  
Notice that the gateways 12, 14, 15, 16 and 17 hops away either don't send ICMP TIME\_EXCEEDED messages or send them with a TTL too small to reach us. 14 - 17 are running the MIT C Gateway code that doesn't send TIME\_EXCEEDEDs.

The silent gateway 12 in the above example may be the result of a bug in the 4.[23]BSD network code (and its derivatives): 4.x (x <= 3) sends an unreachable message using whatever TTL remains in the original datagram. Since, for gateways, the remaining TTL is zero, the ICMP TIME\_EXCEEDED is guaranteed to not make it back to us. The behavior of this bug is slightly more interesting when it appears on the destination system:

```
% traceroute rip.berkeley.edu
 1 helios.ee.lbl.gov (128.3.112.1) 0 ms 0 ms 0 ms
 2 lilac-dmc.Berkeley.EDU (128.32.216.1) 39 ms 19 ms 39 ms
 3 lilac-dmc.Berkeley.EDU (128.32.216.1) 19 ms 39 ms 19 ms
 4 ccngw-ner-cc.Berkeley.EDU (128.32.136.23) 39 ms 40 ms 19 ms
 5 ccn-nerif35.Berkeley.EDU (128.32.168.35) 39 ms 39 ms 39 ms
 6 csgw.Berkeley.EDU (128.32.133.254) 39 ms 59 ms 39 ms
 7 * * *
 8 * * *
 9 * * *
```

Page 3

TRACEROUTE(1M)

TRACEROUTE(1M)

```
10 * * *
11 * * *
12 * * *
13 rip.Berkeley.EDU (128.32.131.22) 59 ms ! 39 ms ! 39 ms !
```

Notice of the 12 ``gateways'' (13 is the final destination), exactly the half of them are ``missing''. In this example, rip, a Sun-3 running Sun OS3.5, is using the TTL from the arriving datagram as the TTL in its ICMP reply. The reply will then time out on the return path, with no notice sent to anyone since ICMP packets aren't sent for ICMP packets, until we probe with a TTL that's at least twice the path length - that is, rip is really only 7 hops away. A reply that returns with a TTL of 1 is a clue this problem exists. Traceroute prints a ``!' after the time if the TTL is <= 1. Since some vendors ship obsolete or nonstandard software, expect to see this problem frequently and/or take care selecting the target host of your probes.

Other possible annotations after the time are !H, !N, !P (got a host, network or protocol unreachable, respectively), !S or !F (source route failed or fragmentation needed - neither of these should ever occur, and the associated gateway is broken if you see one). If almost all the probes result in some kind of unreachable, traceroute will give up and exit.

(ttl=n!) indicates that the TTL value in the ICMP TIME\_EXCEEDED packet that we received was "unexpected". What we expect is that the value will be (some initial value - the number of routers between us). In other words, if the path from hop 5 to us is the same as the path from us to hop 5, we expect to receive a TTL value of (some initial value - 4). Unfortunately, there are several common "initial value"s for ICMP TTLs. The most common are 255, 60, 59, 30, 29. (IRIX, 4.3BSD-tahoe and cisco routers use 255, Proteon routers use either 59 or 29 depending on software release, several other implementations use 60 and 30.) Traceroute checks against all of these, making it hard to detect some "off by one" routing asymmetries. If you want to see all the TTL values in all the packets, use the -l option.

For example,

```
% traceroute -g 10.3.0.5 128.182.0.0
will show the path from the Cambridge Mailbridge to PSC while
```

```
% traceroute -g 192.5.146.4 -g 10.3.0.5 35.0.0.0
```

shows how the Cambridge Mailbridge reaches Merit, by using PSC to reach the Mailbridge.

This program is intended for use in network testing, measurement, and management. It should be used primarily for manual fault isolation. It is unwise to use traceroute during normal operations or from automated scripts due to the load it could impose on the network.

Page 4

TRACEROUTE(1M)

TRACEROUTE(1M)

#### AUTHORS

Van Jacobson, Steve Deering, C. Philip Wood, Tim Seaver, and Ken Adelman.

#### SEE ALSO

netstat(1), ping(1M)



## APPENDIX E. *nslookup* MAN PAGE (UNIX)

NSLOOKUP(1C)

NSLOOKUP(1C)

### NAME

*nslookup* - query Internet name servers interactively

### SYNOPSIS

*nslookup* [ -option ... ] [ host-to-find | - [ server ] ]

### DESCRIPTION

*Nslookup* is a program to query Internet domain name servers. *Nslookup* has two modes: interactive and non-interactive. Interactive mode allows the user to query name servers for information about various hosts and domains or to print a list of hosts in a domain. Non-interactive mode is used to print just the name and requested information for a host or domain.

### ARGUMENTS

Interactive mode is entered in the following cases:

- a) when no arguments are given (the default name server will be used),
- b) when the first argument is a hyphen (-) and the second argument is the host name or Internet address of a name server.

Non-interactive mode is used when the name or Internet address of the host to be looked up is given as the first argument. The optional second argument specifies the host name or address of a name server.

The options listed under the ``set'' command below can be specified in the *.nslookuprc* file in the user's home directory if they are listed one per line. Options can also be specified on the command line if they precede the arguments and are prefixed with a hyphen. For example, to change the default query type to host information, and the initial timeout to 10 seconds, type:

```
nslookup -query=hinfo -timeout=10
```

### INTERACTIVE COMMANDS

Commands may be interrupted at any time by typing a control-C. To exit, type a control-D (EOF) or type exit. The command line length must be less than 256 characters. To treat a built-in command as a host name, precede it with an escape character (\). N.B. an unrecognized command will be interpreted as a host name.

host [server]

Look up information for host using the current default server or using server if specified. If host is an Internet address and the query type is A or PTR, the name of the host is returned. If host is a name and does not have a trailing period, the default domain name is appended to the name. (This behavior depends on the state of the set options domain, srchlist, defname, and search). To look up a host not in the current domain, append a period to the name.

Page 1

**server domain**  
**lserver domain**  
 Change the default server to domain. Lserver uses the initial server to look up information about domain while server uses the current default server. If an authoritative answer can't be found, the names of servers that might have the answer are returned.

**root** Changes the default server to the server for the root of the domain name space. Currently, the host ns.internic.net is used. (This command is a synonym for lserver ns.internic.net.) The name of the root server can be changed with the set root command.

**finger [name] [> filename]**  
**finger [name] [>> filename]**  
 Connects with the finger server on the current host. The current host is defined when a previous lookup for a host was successful and returned address information (see the set querytype=A command). Name is optional. > and >> can be used to redirect output in the usual manner.

**ls [option] domain [> filename]**  
**ls [option] domain [>> filename]**  
 List the information available for domain, optionally creating or appending to filename. The default output contains host names and their Internet addresses. Option can be one of the following:

- t querytype  
     lists all records of the specified type (see querytype below).
- a lists aliases of hosts in the domain. synonym for -t CNAME.
- d lists all records for the domain. synonym for -t ANY.
- h lists CPU and operating system information for the domain.  
     synonym for -t HINFO.
- s lists well-known services of hosts in the domain. synonym for  
     -t WKS.

When output is directed to a file, hash marks are printed for every 50 records received from the server.

**view filename**  
 Sorts and lists the output of previous ls command(s) with more(1).

help

? Prints a brief summary of commands.

exit Exits the program.

set keyword[=value]

This command is used to change state information that affects the lookups. Valid keywords are:

all Prints the current values of the frequently-used options to set. Information about the current default server and host is also printed.

class=value

Change the query class to one of:

IN the Internet class.

CHAOS the Chaos class.

HESIOD the MIT Athena Hesiod class.

ANY wildcard (any of the above).

The class specifies the protocol group of the information.  
(Default = IN, abbreviation = cl)

[no]debug

Turn debugging mode on. A lot more information is printed about the packet sent to the server and the resulting answer.  
(Default = nodebug, abbreviation = [no]deb)

[no]d2

Turn exhaustive debugging mode on. Essentially all fields of every packet are printed.  
(Default = nod2)

domain=name

Change the default domain name to name. The default domain name is appended to a lookup request depending on the state of the defname and search options. The domain search list contains the parents of the default domain if it has at least two components in its name. For example, if the default domain is CC.Berkeley.EDU, the search list is CC.Berkeley.EDU and Berkeley.EDU. Use the set srchlist command to specify a different list. Use the set all command to display the list.  
(Default = value from hostname, /usr/etc/resolv.conf or LOCALDOMAIN, abbreviation = do)

srchlist=name1/name2/...

Change the default domain name to name1 and the domain search list to name1, name2, etc. A maximum of 6 names separated by slashes (/) can be specified. For example,

```
set srchlist=lcs.MIT.EDU/ai.MIT.EDU/MIT.EDU
sets the domain to lcs.MIT.EDU and the search list to the three
names. This command overrides the default domain name and
search list of the set domain command. Use the set all command
to display the list.
(Default = value based on hostname, /usr/etc/resolv.conf or
LOCALDOMAIN, abbreviation = srchl)
```

[no]defname

If set, append the default domain name to a single-component lookup request (i.e., one that does not contain a period). (Default = defname, abbreviation = [no]def)

[no]search

If the lookup request contains at least one period but doesn't end with a trailing period, append the domain names in the domain search list to the request until an answer is received. (Default = search, abbreviation = [no]sea)

port=value

Change the default TCP/UDP name server port to value. (Default = 53, abbreviation = po)

querytype=value

type=value

Change the type of information query to one of:

A	the host's Internet address.
CNAME	the canonical name for an alias.
HINFO	the host CPU and operating system type.
MINFO	the mailbox or mail list information.
MX	the mail exchanger.
NS	the name server for the named zone.
PTR	the host name if the query is an Internet address, otherwise the pointer to other information.
SOA	the domain's ``start-of-authority'' information.
TXT	the text information.

UINFO      the user information.

WKS        the supported well-known services.

Other types (ANY, AXFR, MB, MD, MF, NULL) are described in the RFC-1035 document.

(Default = A, abbreviations = q, ty)

[no]recurse

Tell the name server to query other servers if it does not have the information.

(Default = recurse, abbreviation = [no]rec)

retry=number

Set the number of retries to number. When a reply to a request is not received within a certain amount of time (changed with set timeout), the timeout period is doubled and the request is resent. The retry value controls how many times a request is resent before giving up.

(Default = 4, abbreviation = ret)

root=host

Change the name of the root server to host. This affects the root command.

(Default = ns.internic.net., abbreviation = ro)

timeout=number

Change the initial timeout interval for waiting for a reply to number seconds. Each retry doubles the timeout period.

(Default = 5 seconds, abbreviation = ti)

[no]vc

Always use a virtual circuit when sending requests to the server.

(Default = novc, abbreviation = [no]v)

[no]ignoretc

Ignore packet truncation errors.

(Default = noignoretc, abbreviation = [no]ig)

#### DIAGNOSTICS

If the lookup request was not successful, an error message is printed.

Possible errors are:

Timed out

The server did not respond to a request after a certain amount of time (changed with set timeout=value) and a certain number of retries (changed with set retry=value).

No response from server

No name server is running on the server machine.

## NSLOOKUP(1C)

## NSLOOKUP(1C)

### No records

The server does not have resource records of the current query type for the host, although the host name is valid. The query type is specified with the set querytype command.

### Non-existent domain

The host or domain name does not exist.

### Connection refused

#### Network is unreachable

The connection to the name or finger server could not be made at the current time. This error commonly occurs with ls and finger requests.

### Server failure

The name server found an internal inconsistency in its database and could not return a valid answer.

### Refused

The name server refused to service the request.

### Format error

The name server found that the request packet was not in the proper format. It may indicate an error in nslookup.

## FILES

/usr/etc/resolv.conf	initial domain name and name server addresses.
\$HOME/.nslookuprc	user's initial options.
/usr/bsd/nslookup.help	summary of commands.

## ENVIRONMENT

HOSTALIASES	file containing host aliases.
LOCALDOMAIN	overrides default domain.

## SEE ALSO

resolver(3), resolver(4), named(1M),  
RFC-1034 ``Domain Names - Concepts and Facilities''  
RFC-1035 ``Domain Names - Implementation and Specification''

## APPENDIX F. *ttcp* MAN PAGE (UNIX)

TTCP(1)

TTCP(1)

### NAME

*ttcp* - test TCP and UDP performance

### SYNOPSIS

```
ttcp -t [-u] [-s] [-p port] [-l buflen] [-b size] [-n numbufs] [-A align]
[-O offset] [-f format] [-D] [-v] host [<in]
ttcp -r [-u] [-s] [-p port] [-l buflen] [-b size] [-A align] [-O offset]
[-f format] [-B] [-T] [-v] [>out]
```

### DESCRIPTION

*Ttcp* times the transmission and reception of data between two systems using the UDP or TCP protocols. It differs from common ``blast'' tests, which tend to measure the remote *inetd* as much as the network performance, and which usually do not allow measurements at the remote end of a UDP transmission.

For testing, the transmitter should be started with *-t* and *-s* after the receiver has been started with *-r* and *-s*. Tests lasting at least tens of seconds should be used to obtain accurate measurements. Graphical presentations of throughput versus buffer size for buffers ranging from tens of bytes to several ``pages'' can illuminate bottlenecks.

*Ttcp* can also be used as a ``network pipe'' for moving directory hierarchies between systems when routing problems exist or when the use of other mechanisms is undesirable. For example, on the destination machine, use:

```
ttcp -r -B | tar xvpf -
```

and on the source machine:

```
tar cf - directory | ttcp -t dest_machine
```

Additional intermediate machines can be included by:

```
ttcp -r | ttcp -t next_machine
```

### OPTIONS

<i>-t</i>	Transmit mode.
<i>-r</i>	Receive mode.
<i>-u</i>	Use UDP instead of TCP.
<i>-s</i>	If transmitting, source a data pattern to network; if receiving, sink (discard) the data. Without the <i>-s</i> option, the default is to transmit data from <i>stdin</i> or print the received data to <i>stdout</i> .

Page 1

## TTCP(1)

## TTCP(1)

-l length Length of buffers in bytes (default 8192). For UDP, this value is the number of data bytes in each packet. The system limits the maximum UDP packet length. This limit can be changed with the -b option.

When testing UDP performance, it is important to set the packet size to be less than or equal to the maximum transmission unit of the media. Otherwise, IP fragmentation will distort the test. For Ethernet, set the length to 1508 bytes.

-b size Set size of socket buffer. The default varies from system to system. This parameter affects the maximum UDP packet length. It may not be possible to set this parameter on some systems (for example, 4.2BSD).

-n numbufs Number of source buffers transmitted (default 2048).

-p port Port number to send to or listen on (default 2000). On some systems, this port may be allocated to another network daemon.

-D If transmitting using TCP, do not buffer data when sending (sets the TCP\_NODELAY socket option). It may not be possible to set this parameter on some systems (for example, 4.2BSD).

-B When receiving data, output only full blocks, using the block size specified by -l. This option is useful for programs, such as tar(1), that require complete blocks.

-A align Align the start of buffers to this modulus (default 16384).

-O offset Align the start of buffers to this offset (default 0). For example, ``-A8192 -O1'' causes buffers to start at the second byte of an 8192-byte page.

-f format Specify, using one of the following characters, the format of the throughput rates as kilobits/sec ('k'), kilobytes/sec ('K'), megabits/sec ('m'), megabytes/sec ('M'), gigabits/sec ('g'), or gigabytes/sec ('G'). The default is 'K'.

-T ``Touch'' the data as they are read in order to measure cache effects.

-v Verbose: print more statistics.

-d Debug: set the SO\_DEBUG socket option.

## SEE ALSO

ping(1M), traceroute(1M), netsnoop(1M)



## APPENDIX G. *tcpdump* MAN PAGE

TCPDUMP(1)

UNIX System V (20 Jun 1994)

TCPDUMP(1)

### NAME

*tcpdump* - dump traffic on a network

### SYNOPSIS

```
tcpdump [ -deflnNOPqStvx ] [ -c count ] [ -F file ]  
        [ -i interface ] [ -r file ] [ -s snaplen ]  
        [ -w file ] expression
```

### DESCRIPTION

*Tcpdump* prints out the headers of packets on a network interface that match the boolean expression. Under SunOS: You must be root to invoke *tcpdump* or it must be installed *setuid* to root. Under Ultrix: Any user can invoke *tcpdump* once the super-user has enabled promiscuous-mode operation using *pfconfig*(8). Under BSD: Access is controlled by the permissions on */dev/bpf0*, etc.

### OPTIONS

- c Exit after receiving count packets.
- d Dump the compiled packet-matching code to standard output and stop.
- e Print the link-level header on each dump line.
- f Print 'foreign' internet addresses numerically rather than symbolically (this option is intended to get around serious brain damage in Sun's yp server - usually it hangs forever translating non-local internet numbers).
- F Use file as input for the filter expression. An additional expression given on the command line is ignored.
- i Listen on interface. If unspecified, *tcpdump* searches the system interface list for the lowest numbered, configured up interface (excluding loopback). Ties are broken by choosing the earliest match.
- l Make stdout line buffered. Useful if you want to see the data while capturing it. E.g.,  
``*tcpdump* -l | tee dat'' or ``*tcpdump* -l > dat & tail -f dat''.
- n Don't convert addresses (i.e., host addresses, port numbers, etc.) to names.
- N Don't print domain name qualification of host names. E.g., if you give this flag then *tcpdump* will print ``nic'' instead of ``nic.ddn.mil''.

- O Do not run the packet-matching code optimizer. This is useful only if you suspect a bug in the optimizer.
- p Don't put the interface into promiscuous mode. Note that the interface might be in promiscuous for some other reason; hence, '-p' cannot be used as an abbreviation for 'ether host {localhost} or broadcast'.
- q Quick (quiet?) output. Print less protocol information so output lines are shorter.
- r Read packets from file (which was created with the -w option). Standard input is used if file is '-'.
- s Snarf snaplen bytes of data from each packet rather than the default of 68 (with NIT, the minimum is actually 96). 68 bytes is adequate for IP, ICMP, TCP and UDP but may truncate protocol information from name server and NFS packets (see below). Packets truncated because of a limited snapshot are indicated in the output with '[|proto]', where proto is the name of the protocol level at which the truncation has occurred. Note that taking larger snapshots both increases the amount of time it takes to process packets and, effectively, decreases the amount of packet buffering. This may cause packets to be lost. You should limit snaplen to the smallest number that will capture the protocol information you're interested in.
- S Print absolute, rather than relative, TCP sequence numbers.
- t Don't print a timestamp on each dump line.
- tt Print an unformatted timestamp on each dump line.
- v (Slightly more) verbose output. For example, the time to live and type of service information in an IP packet is printed.
- vv Even more verbose output. For example, additional fields are printed from NFS reply packets.
- w Write the raw packets to file rather than parsing and printing them out. They can later be printed with the -r option. Standard output is used if file is '-'.
- x Print each packet (minus its link level header) in hex. The smaller of the entire packet or snaplen bytes will be printed.

**expression**

selects which packets will be dumped. If no expression is given, all packets on the net will be dumped. Otherwise, only packets for which expression is 'true' will be dumped.

The expression consists of one or more primitives. Primitives usually consist of an id (name or number) preceded by one or more qualifiers. There are three different kinds of qualifier:

**type** qualifiers say what kind of thing the id name or number refers to. Possible types are host, net and port. E.g., 'host foo', 'net 128.3', 'port 20'. If there is no type qualifier, host is assumed.

**dir** qualifiers specify a particular transfer direction to and/or from id. Possible directions are src, dst, src or dst and src and dst. E.g., 'src foo', 'dst net 128.3', 'src or dst port ftp-data'. If there is no dir qualifier, src or dst is assumed.

**proto**

qualifiers restrict the match to a particular protocol. Possible protos are: ether, fddi, ip, arp, rarp, decnet, lat, mopr, mopdl, tcp and udp. E.g., 'ether src foo', 'arp net 128.3', 'tcp port 21'. If there is no proto qualifier, all protocols consistent with the type are assumed. E.g., 'src foo' means '(ip or arp or rarp) src foo' (except the latter is not legal syntax), 'net bar' means '(ip or arp or rarp) net bar' and 'port 53' means '(tcp or udp) port 53'.

['fddi' is actually an alias for 'ether'; the parser treats them identically as meaning 'the data link level used on the specified network interface.' FDDI headers contain Ethernet-like source and destination addresses, and often contain Ethernet-like packet types, so you can filter on these FDDI fields just as with the analogous Ethernet fields. FDDI headers also contain other fields, but you cannot name them explicitly in a filter expression.]

In addition to the above, there are some special 'primitive' keywords that don't follow the pattern: gateway, broadcast, less, greater and arithmetic expressions. All of these are described below.

More complex filter expressions are built up by using the words and, or and not to combine primitives. E.g.,

'host foo and not port ftp and not port ftp-data'. To save typing, identical qualifier lists can be omitted. E.g., 'tcp dst port ftp or ftp-data or domain' is exactly the same as 'tcp dst port ftp or tcp dst port ftp-data or tcp dst port domain'.

Allowable primitives are:

dst host host

True if the IP destination field of the packet is host, which may be either an address or a name.

src host host

True if the IP source field of the packet is host.

host host

True if either the IP source or destination of the packet is host. Any of the above host expressions can be prepended with the keywords, ip, arp, or rarp as in:

ip host host

which is equivalent to:

ether proto \ip and host host

If host is a name with multiple IP addresses, each address will be checked for a match.

ether dst ehost

True if the ethernet destination address is ehost. Ehost may be either a name from /etc/ethers or a number (see ethers(3N) for numeric format).

ether src ehost

True if the ethernet source address is ehost.

ether host ehost

True if either the ethernet source or destination address is ehost.

gateway host

True if the packet used host as a gateway. I.e., the ethernet source or destination address was host but neither the IP source nor the IP destination was host. Host must be a name and must be found in both /etc/hosts and /etc/ethers. (An equivalent expression is

ether host ehost and not host host

which can be used with either names or numbers for host / ehost.)

dst net net

True if the IP destination address of the packet has a network number of net, which may be either

an address or a name.

src net net

True if the IP source address of the packet has a network number of net.

net net

True if either the IP source or destination address of the packet has a network number of net.

dst port port

True if the packet is ip/tcp or ip/udp and has a destination port value of port. The port can be a number or a name used in /etc/services (see tcp(4P) and udp(4P)). If a name is used, both the port number and protocol are checked. If a number or ambiguous name is used, only the port number is checked (e.g., dst port 513 will print both tcp/login traffic and udp/who traffic, and port domain will print both tcp/domain and udp/domain traffic).

src port port

True if the packet has a source port value of port.

port port

True if either the source or destination port of the packet is port. Any of the above port expressions can be prepended with the keywords, tcp or udp, as in:

tcp src port port

which matches only tcp packets.

less length

True if the packet has a length less than or equal to length. This is equivalent to:

len <= length.

greater length

True if the packet has a length greater than or equal to length. This is equivalent to:

len >= length.

ip proto protocol

True if the packet is an ip packet (see ip(4P)) of protocol type protocol. Protocol can be a number or one of the names icmp, udp, nd, or tcp. Note that the identifiers tcp, udp, and icmp are also keywords and must be escaped via backslash (\), which is \\ in the C-shell.

**ether broadcast**

True if the packet is an ethernet broadcast packet. The ether keyword is optional.

**ip broadcast**

True if the packet is an IP broadcast packet. It checks for both the all-zeroes and all-ones broadcast conventions, and looks up the local subnet mask.

**ether multicast**

True if the packet is an ethernet multicast packet. The ether keyword is optional. This is shorthand for 'ether[0] & 1 != 0'.

**ip multicast**

True if the packet is an IP multicast packet.

**ether proto protocol**

True if the packet is of ether type protocol. Protocol can be a number or a name like ip, arp, or rarp. Note these identifiers are also keywords and must be escaped via backslash (\). [In the case of FDDI (e.g., 'fddi protocol arp'), the protocol identification comes from the 802.2 Logical Link Control (LLC) header, which is usually layered on top of the FDDI header. tcpdump assumes, when filtering on the protocol identifier, that all FDDI packets include an LLC header, and that the LLC header is in so-called SNAP format.]

**decnet src host**

True if the DECNET source address is host, which may be an address of the form '10.123', or a DECNET host name. [DECNET host name support is only available on Ultrix systems that are configured to run DECNET.]

**decnet dst host**

True if the DECNET destination address is host.

**decnet host host**

True if either the DECNET source or destination address is host.

**ip, arp, rarp, decnet**

Abbreviations for:

ether proto p

where p is one of the above protocols.

**lat, moprc, mopdl**

Abbreviations for:

ether proto p

where p is one of the above protocols. Note that tcpdump does not currently know how to parse these protocols.

tcp, udp, icmp

Abbreviations for:

ip proto p

where p is one of the above protocols.

expr relop expr

True if the relation holds, where relop is one of >, <, >=, <=, =, !=, and expr is an arithmetic expression composed of integer constants (expressed in standard C syntax), the normal binary operators [+ , - , \* , / , & , |], a length operator, and special packet data accessors. To access data inside the packet, use the following syntax:

proto [ expr : size ]

Proto is one of ether, fddi, ip, arp, rarp, tcp, udp, or icmp, and indicates the protocol layer for the index operation. The byte offset, relative to the indicated protocol layer, is given by expr. Size is optional and indicates the number of bytes in the field of interest; it can be either one, two, or four, and defaults to one. The length operator, indicated by the keyword len, gives the length of the packet.

For example, 'ether[0] & 1 != 0' catches all multicast traffic. The expression 'ip[0] & 0xf != 5' catches all IP packets with options. The expression 'ip[6:2] & 0x1fff = 0' catches only unfragmented datagrams and frag zero of fragmented datagrams. This check is implicitly applied to the tcp and udp index operations. For instance, tcp[0] always means the first byte of the TCP header, and never means the first byte of an intervening fragment.

Primitives may be combined using:

A parenthesized group of primitives and operators (parentheses are special to the Shell and must be escaped).

Negation ('!' or 'not').

Concatenation ('&&' or 'and').

Alternation ('||' or 'or').

Negation has highest precedence. Alternation and concatenation have equal precedence and associate left to right. Note that explicit and tokens, not juxtaposition, are now required for concatenation.

If an identifier is given without a keyword, the most recent keyword is assumed. For example,

```
not host vs and ace
is short for
not host vs and host ace
which should not be confused with
not ( host vs or ace )
```

Expression arguments can be passed to tcpdump as either a single argument or as multiple arguments, whichever is more convenient. Generally, if the expression contains Shell metacharacters, it is easier to pass it as a single, quoted argument. Multiple arguments are concatenated with spaces before being parsed.

#### EXAMPLES

To print all packets arriving at or departing from sundown:  
tcpdump host sundown

To print traffic between helios and either hot or ace:  
tcpdump host helios and \( hot or ace \)

To print all IP packets between ace and any host except helios:  
tcpdump ip host ace and not helios

To print all traffic between local hosts and hosts at Berkeley:  
tcpdump net ucb-ether

To print all ftp traffic through internet gateway snup:  
(note that the expression is quoted to prevent the shell from (mis-)interpreting the parentheses):  
tcpdump 'gateway snup and (port ftp or ftp-data)'

To print traffic neither sourced from nor destined for local hosts (if you gateway to one other net, this stuff should never make it onto your local net).  
tcpdump ip and not net localnet

To print the start and end packets (the SYN and FIN packets) of each TCP conversation that involves a non-local host.  
tcpdump 'tcp[13] & 3 != 0 and not src and dst net localnet'

To print IP packets longer than 576 bytes sent through



gateway snup:

```
tcpdump 'gateway snup and ip[2:2] > 576'
```

To print IP broadcast or multicast packets that were not sent via ethernet broadcast or multicast:

```
tcpdump 'ether[0] & 1 = 0 and ip[16] >= 224'
```

To print all ICMP packets that are not echo requests/replies (i.e., not ping packets):

```
tcpdump 'icmp[0] != 8 and icmp[0] != 0'
```

#### OUTPUT FORMAT

The output of tcpdump is protocol dependent. The following gives a brief description and examples of most of the formats.

##### Link Level Headers

If the '-e' option is given, the link level header is printed out. On ethernets, the source and destination addresses, protocol, and packet length are printed.

On FDDI networks, the '-e' option causes tcpdump to print the 'frame control' field, the source and destination addresses, and the packet length. (The 'frame control' field governs the interpretation of the rest of the packet. Normal packets (such as those containing IP datagrams) are 'async' packets, with a priority value between 0 and 7; for example, 'async4'. Such packets are assumed to contain an 802.2 Logical Link Control (LLC) packet; the LLC header is printed if it is not an ISO datagram or a so-called SNAP packet.

(N.B.: The following description assumes familiarity with the SLIP compression algorithm described in RFC-1144.)

On SLIP links, a direction indicator ('I' for inbound, 'O' for outbound), packet type, and compression information are printed out. The packet type is printed first. The three types are ip, utcp, and ctcp. No further link information is printed for ip packets. For TCP packets, the connection identifier is printed following the type. If the packet is compressed, its encoded header is printed out. The special cases are printed out as \*S+n and \*SA+n, where n is the amount by which the sequence number (or sequence number and ack) has changed. If it is not a special case, zero or more changes are printed. A change is indicated by U (urgent pointer), W (window), A (ack), S (sequence number), and I (packet ID), followed by a delta (+n or -n), or a new value (=n). Finally, the amount of data in the packet and compressed header length are printed.

For example, the following line shows an outbound compressed TCP packet, with an implicit connection identifier; the ack has changed by 6, the sequence number by 49, and the packet ID by 6; there are 3 bytes of data and 6 bytes of compressed header:

```
0 tcp * A+6 S+49 I+6 3 (6)
```

#### ARP/RARP Packets

Arp/rarp output shows the type of request and its arguments. The format is intended to be self explanatory. Here is a short sample taken from the start of an 'rlogin' from host rtsg to host csam:

```
arp who-has csam tell rtsg
arp reply csam is-at CSAM
```

The first line says that rtsg sent an arp packet asking for the ethernet address of internet host csam. Csam replies with its ethernet address (in this example, ethernet addresses are in caps and internet addresses in lower case).

This would look less redundant if we had done tcpdump -n:

```
arp who-has 128.3.254.6 tell 128.3.254.68
arp reply 128.3.254.6 is-at 02:07:01:00:01:c4
```

If we had done tcpdump -e, the fact that the first packet is broadcast and the second is point-to-point would be visible:

```
RTSG Broadcast 0806 64: arp who-has csam tell rtsg
CSAM RTSG 0806 64: arp reply csam is-at CSAM
```

For the first packet this says the ethernet source address is RTSG, the destination is the broadcast address, the type field contained hex 0806 (type ETHER\_ARP) and the total length was 64 bytes.

#### TCP Packets

(N.B.:The following description assumes familiarity with the TCP protocol described in RFC-793. If you are not familiar with the protocol, neither this description nor tcpdump will be of much use to you.)

The general format of a tcp protocol line is:

```
src > dst: flags data-seqno ack window urgent options
```

Src and dst are the source and destination IP addresses and ports. Flags are some combination of S (SYN), F (FIN), P (PUSH) or R (RST) or a single '.' (no flags). Data-seqno describes the portion of sequence space covered by the data in this packet (see example below). Ack is sequence number of the next data expected the other direction on this

connection. Window is the number of bytes of receive buffer space available the other direction on this connection. Urg indicates there is 'urgent' data in the packet. Options are tcp options enclosed in angle brackets (e.g., <mss 1024>).

Src, dst and flags are always present. The other fields depend on the contents of the packet's tcp protocol header and are output only if appropriate.

Here is the opening portion of an rlogin from host rtsg to host csam.

```
rtsg.1023 > csam.login: S 768512:768512(0) win 4096 <mss 1024>
csam.login > rtsg.1023: S 947648:947648(0) ack 768513 win 4096
<mss 1024>
rtsg.1023 > csam.login: . ack 1 win 4096
rtsg.1023 > csam.login: P 1:2(1) ack 1 win 4096
csam.login > rtsg.1023: . ack 2 win 4096
rtsg.1023 > csam.login: P 2:21(19) ack 1 win 4096
csam.login > rtsg.1023: P 1:2(1) ack 21 win 4077
csam.login > rtsg.1023: P 2:3(1) ack 21 win 4077 urg 1
csam.login > rtsg.1023: P 3:4(1) ack 21 win 4077 urg 1
```

The first line says that tcp port 1023 on rtsg sent a packet to port login on csam. The S indicates that the SYN flag was set. The packet sequence number was 768512 and it contained no data. (The notation is 'first:last(nbytes)' which means 'sequence numbers first up to but not including last which is nbytes bytes of user data'.) There was no piggy-backed ack, the available receive window was 4096 bytes and there was a max-segment-size option requesting an mss of 1024 bytes.

Csam replies with a similar packet except it includes a piggy-backed ack for rtsg's SYN. Rtsg then acks csam's SYN. The '.' means no flags were set. The packet contained no data so there is no data sequence number. Note that the ack sequence number is a small integer (1). The first time tcpdump sees a tcp 'conversation', it prints the sequence number from the packet. On subsequent packets of the conversation, the difference between the current packet's sequence number and this initial sequence number is printed. This means that sequence numbers after the first can be interpreted as relative byte positions in the conversation's data stream (with the first data byte each direction being '1'). '-S' will override this feature, causing the original sequence numbers to be output.

On the 6th line, rtsg sends csam 19 bytes of data (bytes 2 through 20 in the rtsg -> csam side of the conversation). The PUSH flag is set in the packet. On the 7th line, csam says it's received data sent by rtsg up to but not including byte 21. Most of this data is apparently sitting in the socket buffer since csam's receive window has gotten 19

bytes smaller. Csam also sends one byte of data to rtsg in this packet. On the 8th and 9th lines, csam sends two bytes of urgent, pushed data to rtsg.

#### UDP Packets

UDP format is illustrated by this rwho packet:

```
actinide.who > broadcast.who: udp 84
```

This says that port who on host actinide sent a udp datagram to port who on host broadcast, the Internet broadcast address. The packet contained 84 bytes of user data.

Some UDP services are recognized (from the source or destination port number) and the higher level protocol information printed. In particular, Domain Name service requests (RFC-1034/1035) and Sun RPC calls (RFC-1050) to NFS.

#### UDP Name Server Requests

(N.B.:The following description assumes familiarity with the Domain Service protocol described in RFC-1035. If you are not familiar with the protocol, the following description will appear to be written in greek.)

Name server requests are formatted as

```
src > dst: id op? flags qtype qclass name (len)
h2opolo.1538 > helios.domain: 3+ A? ucbvax.berkeley.edu.
(37)
```

Host h2opolo asked the domain server on helios for an address record (qtype=A) associated with the name ucbvax.berkeley.edu. The query id was '3'. The '+' indicates the recursion desired flag was set. The query length was 37 bytes, not including the UDP and IP protocol headers. The query operation was the normal one, Query, so the op field was omitted. If the op had been anything else, it would have been printed between the '3' and the '+'. Similarly, the qclass was the normal one, C\_IN, and omitted. Any other qclass would have been printed immediately after the 'A'.

A few anomalies are checked and may result in extra fields enclosed in square brackets: If a query contains an answer, name server or authority section, amount, nscount, or arcount are printed as '[na]', '[nn]' or '[nau]' where n is the appropriate count. If any of the response bits are set (AA, RA or rcode) or any of the 'must be zero' bits are set in bytes two and three, '[b2&3=x]' is printed, where x is the hex value of header bytes two and three.

## UDP Name Server Responses

Name server responses are formatted as

```
src > dst: id op rcode flags a/n/au type class data (len)
helios.domain > h2opolo.1538: 3 3/3/7 A 128.32.137.3 (273)
helios.domain > h2opolo.1537: 2 NXDomain* 0/1/0 (97)
```

In the first example, helios responds to query id 3 from h2opolo with 3 answer records, 3 name server records and 7 authority records. The first answer record is type A (address) and its data is internet address 128.32.137.3. The total size of the response was 273 bytes, excluding UDP and IP headers. The op (Query) and response code (NoError) were omitted, as was the class (C\_IN) of the A record.

In the second example, helios responds to query 2 with a response code of non-existent domain (NXDomain) with no answers, one name server and no authority records. The '\*' indicates that the authoritative answer bit was set. Since there were no answers, no type, class or data were printed.

Other flag characters that might appear are '-' (recursion available, RA, not set) and '|' (truncated message, TC, set). If the 'question' section doesn't contain exactly one entry, '[nq]' is printed.

Note that name server requests and responses tend to be large and the default snaplen of 96 bytes may not capture enough of the packet to print. Use the -s flag to increase the snaplen if you need to seriously investigate name server traffic. '-s 128' has worked well for me.

## NFS Requests and Replies

Sun NFS (Network File System) requests and replies are printed as:

```
src.xid > dst.nfs: len op args
src.nfs > dst.xid: reply stat len op results

sushi.6709 > wr1.nfs: 112 readlink fh 21,24/10.73165
wr1.nfs > sushi.6709: reply ok 40 readlink "../var"
sushi.201b > wr1.nfs:
144 lookup fh 9,74/4096.6878 "xcolors"
wr1.nfs > sushi.201b:
reply ok 128 lookup fh 9,74/4134.3150
```

In the first line, host sushi sends a transaction with id 6709 to wr1 (note that the number following the src host is a transaction id, not the source port). The request was 112

bytes, excluding the UDP and IP headers. The operation was a readlink (read symbolic link) on file handle (fh) 21,24/10.731657119. (If one is lucky, as in this case, the file handle can be interpreted as a major,minor device number pair, followed by the inode number and generation number.) Wrl replies 'ok' with the contents of the link.

In the third line, sushi asks wr1 to lookup the name 'xcolors' in directory file 9,74/4096.6878. Note that the data printed depends on the operation type. The format is intended to be self explanatory if read in conjunction with an NFS protocol spec.

If the -v (verbose) flag is given, additional information is printed. For example:

```
sushi.1372a > wr1.nfs:
148 read fh 21,11/12.195 8192 bytes @ 24576
wr1.nfs > sushi.1372a:
reply ok 1472 read REG 100664 ids 417/0 sz 29388
```

(-v also prints the IP header TTL, ID, and fragmentation fields, which have been omitted from this example.) In the first line, sushi asks wr1 to read 8192 bytes from file 21,11/12.195, at byte offset 24576. Wrl replies 'ok'; the packet shown on the second line is the first fragment of the reply, and hence is only 1472 bytes long (the other bytes will follow in subsequent fragments, but these fragments do not have NFS or even UDP headers and so might not be printed, depending on the filter expression used). Because the -v flag is given, some of the file attributes (which are returned in addition to the file data) are printed: the file type ('REG', for regular file), the file mode (in octal), the uid and gid, and the file size.

If the -v flag is given more than once, even more details are printed.

Note that NFS requests are very large and much of the detail won't be printed unless snaplen is increased. Try using '-s 192' to watch NFS traffic.

NFS reply packets do not explicitly identify the RPC operation. Instead, tcpdump keeps track of 'recent' requests, and matches them to the replies using the transaction ID. If a reply does not closely follow the corresponding request, it might not be parseable.

KIP Appletalk (DDP in UDP)

Appletalk DDP packets encapsulated in UDP datagrams are de-encapsulated and dumped as DDP packets (i.e., all the UDP header information is discarded). The file /etc/atalk.names is used to translate appletalk net and node numbers to names. Lines in this file have the form

```

number    name
1.254      ether
16.1       icسد-net
1.254.110 ace
```

The first two lines give the names of appletalk networks. The third line gives the name of a particular host (a host is distinguished from a net by the 3rd octet in the number - a net number must have two octets and a host number must have three octets.) The number and name should be separated by whitespace (blanks or tabs). The /etc/atalk.names file may contain blank lines or comment lines (lines starting with a '#').

Appletalk addresses are printed in the form  
net.host.port

```

144.1.209.2 > icسد-net.112.220
office.2 > icسد-net.112.220
jssmag.149.235 > icسد-net.2
```

(If the /etc/atalk.names doesn't exist or doesn't contain an entry for some appletalk host/net number, addresses are printed in numeric form.) In the first example, NBP (DDP port 2) on net 144.1 node 209 is sending to whatever is listening on port 220 of net icسد node 112. The second line is the same except the full name of the source node is known ('office'). The third line is a send from port 235 on net jssmag node 149 to broadcast on the icسد-net NBP port (note that the broadcast address (255) is indicated by a net name with no host number - for this reason it's a good idea to keep node names and net names distinct in /etc/atalk.names).

NBP (name binding protocol) and ATP (Appletalk transaction protocol) packets have their contents interpreted. Other protocols just dump the protocol name (or number if no name is registered for the protocol) and packet size.

```

NBP packets are formatted like the following examples:
icسد-net.112.220 > jssmag.2: nbp-lkup 190: "=:LaserWriter@*"
jssmag.209.2 > icسد-net.112.220: nbp-reply 190
      : "RM1140:LaserWriter@" 250
techpit.2 > icسد-net.112.220: nbp-reply 190:
      "techpit:LaserWriter@" 186
```

The first line is a name lookup request for laserwriters sent by net icسد host 112 and broadcast on net jssmag. The nbp id for the lookup is 190. The second line shows a reply

for this request (note that it has the same id) from host jssmag.209 saying that it has a laserwriter resource named "RM1140" registered on port 250. The third line is another reply to the same request saying host techpit has laserwriter "techpit" registered on port 186.

ATP packet formatting is demonstrated by the following example:

```
jssmag.209.165 > helios.132: atp-req 12266<0-7> 0xae030001
helios.132 > jssmag.209.165: atp-resp 12266:0 (512) 0xae040000
helios.132 > jssmag.209.165: atp-resp 12266:1 (512) 0xae040000
helios.132 > jssmag.209.165: atp-resp 12266:2 (512) 0xae040000
helios.132 > jssmag.209.165: atp-resp 12266:3 (512) 0xae040000
helios.132 > jssmag.209.165: atp-resp 12266:4 (512) 0xae040000
helios.132 > jssmag.209.165: atp-resp 12266:5 (512) 0xae040000
helios.132 > jssmag.209.165: atp-resp 12266:6 (512) 0xae040000
helios.132 > jssmag.209.165: atp-resp*12266:7 (512) 0xae040000
jssmag.209.165 > helios.132: atp-req 12266<3,5> 0xae030001
helios.132 > jssmag.209.165: atp-resp 12266:3 (512) 0xae040000
helios.132 > jssmag.209.165: atp-resp 12266:5 (512) 0xae040000
jssmag.209.165 > helios.132: atp-rel 12266<0-7> 0xae030001
jssmag.209.133 > helios.132: atp-req* 12267<0-7> 0xae030002
```

Jssmag.209 initiates transaction id 12266 with host helios by requesting up to 8 packets (the '<0-7>'). The hex number at the end of the line is the value of the 'userdata' field in the request.

Helios responds with 8 512-byte packets. The ':digit' following the transaction id gives the packet sequence number in the transaction and the number in parens is the amount of data in the packet, excluding the atp header. The '\*' on packet 7 indicates that the EOM bit was set.

Jssmag.209 then requests that packets 3 & 5 be retransmitted. Helios resends them then jssmag.209 releases the transaction. Finally, jssmag.209 initiates the next request. The '\*' on the request indicates that XO ('exactly once') was not set.

#### IP Fragmentation

Fragmented Internet datagrams are printed as

```
(frag id:size@offset+)
```

```
(frag id:size@offset)
```

(The first form indicates there are more fragments. The second indicates this is the last fragment.)

Id is the fragment id. Size is the fragment size (in bytes) excluding the IP header. Offset is this fragment's offset



(in bytes) in the original datagram.

The fragment information is output for each fragment. The first fragment contains the higher level protocol header and the frag info is printed after the protocol info. Fragments after the first contain no higher level protocol header and the frag info is printed after the source and destination addresses. For example, here is part of an ftp from arizona.edu to lbl-rtsg.arpa over a CSNET connection that doesn't appear to handle 576 byte datagrams:

```
arizona.ftp-data > rtsg.1170: . 1024:1332(308) ack 1 win
4096 (frag 595a:328@0+)
arizona > rtsg: (frag 595a:204@328)
rtsg.1170 > arizona.ftp-data: . ack 1536 win 2560
```

There are a couple of things to note here: First, addresses in the 2nd line don't include port numbers. This is because the TCP protocol information is all in the first fragment and we have no idea what the port or sequence numbers are when we print the later fragments. Second, the tcp sequence information in the first line is printed as if there were 308 bytes of user data when, in fact, there are 512 bytes (308 in the first frag and 204 in the second). If you are looking for holes in the sequence space or trying to match up acks with packets, this can fool you.

A packet with the IP don't fragment flag is marked with a trailing (DF).

#### Timestamps

By default, all output lines are preceded by a timestamp. The timestamp is the current clock time in the form

```
hh:mm:ss.frac
```

and is as accurate as the kernel's clock (e.g., \_10ms on a Sun-3). The timestamp reflects the time the kernel first saw the packet. No attempt is made to account for the time lag between when the ethernet interface removed the packet from the wire and when the kernel serviced the 'new packet' interrupt (of course, with Sun's lousy clock resolution this time lag is negligible.)

#### SEE ALSO

traffic(1C), nit(4P), bpf(4)

#### AUTHORS

Van Jacobson (van@helios.ee.lbl.gov), Craig Leres (leres@helios.ee.lbl.gov) and Steven McCanne (mccanne@helios.ee.lbl.gov), all of Lawrence Berkeley Laboratory, University of California, Berkeley, CA.

#### BUGS

The clock resolution on most Suns is pathetic (20ms). If

you want to use the timestamp to generate some of the important performance distributions (like packet interarrival time) it's best to watch something that generates packets slowly (like an Arpanet gateway or a MicroVax running VMS).

NIT doesn't let you watch your own outbound traffic, BPF will. We recommend that you use the latter.

tcpdump for Ultrix requires Ultrix version 4.0 or later; the kernel has to have been built with the packetfilter pseudo-device driver (see packetfilter(4)). In order to watch either your own outbound or inbound traffic, you will need to use Ultrix version 4.2 or later, and you will have to have used the pfconfig(8) command to enable ``copyall'' mode.

Under SunOS 4.1, the packet capture code (or Streams NIT) is not what you'd call efficient. Don't plan on doing much with your Sun while you're monitoring a busy network.

On Sun systems prior to release 3.2, NIT is very buggy. If run on an old system, tcpdump may crash the machine.

Some attempt should be made to reassemble IP fragments or, at least to compute the right length for the higher level protocol.

Name server inverse queries are not dumped correctly: The (empty) question section is printed rather than real query in the answer section. Some believe that inverse queries are themselves a bug and prefer to fix the program generating them rather than tcpdump.

Apple Ethertalk DDP packets could be dumped as easily as KIP DDP packets but aren't. Even if we were inclined to do anything to promote the use of Ethertalk (we aren't), LBL doesn't allow Ethertalk on any of its networks so we'd would have no way of testing this code.

A packet trace that crosses a daylight savings time change will give skewed time stamps (the time change is ignored).

Filters expressions that manipulate FDDI headers assume that all FDDI packets are encapsulated Ethernet packets. This is true for IP, ARP, and DECNET Phase IV, but is not true for protocols such as ISO CLNS. Therefore, the filter may inadvertently accept certain packets that do not properly match the filter expression.

## APPENDIX H. *mrinfo* MAN PAGE

MRINFO(8)

UNIX System V

MRINFO(8)

### NAME

*mrinfo* - Displays configuration info from a multicast router

### SYNOPSIS

```
/usr/sbin/mrinfo [ -d debug_level ] [ -r retry_count ] [ -t
timeout_count ] multicast_router
```

### DESCRIPTION

*mrinfo* attempts to display the configuration information from the multicast router *multicast\_router*.

*mrinfo* uses the ASK\_NEIGHBORS IGMP message to the specified multicast router. If this multicast router responds, the version number and a list of their neighboring multicast router addresses is part of that response. If the responding router has a recent multicast version number, then *mrinfo* requests additional information such as metrics, thresholds, and flags from the multicast router. Once the specified multicast router responds, the configuration is displayed to the standard output.

### INVOCATION

"-d" option sets the debug level. When the debug level is greater than the default value of 0, addition debugging messages are printed. Regardless of the debug level, an error condition, will always write an error message and will cause *mrinfo* to terminate. Non-zero debug levels have the following effects:

#### level 1

packet warnings are printed to stderr.

#### level 2

all level 1 messages plus notifications down networks are printed to stderr.

#### level 3

all level 2 messages plus notifications of all packet timeouts are printed to stderr.

"-r *retry\_count*" sets the neighbor query retry limit. Default is 3 retry.

"-t *timeout\_count*" sets the number of seconds to wait for a neighbor query reply. Default timeout is 4 seconds.

### SAMPLE OUTPUT

```
mrinfo mbone.phony.dom.net
127.148.176.10 (mbone.phony.dom.net) [version 3.3]:
127.148.176.10 -> 0.0.0.0 (?) [1/1/querier]
127.148.176.10 -> 127.0.8.4 (mbone2.phony.dom.net) [1/45/tunnel]
```

```
127.148.176.10 -> 105.1.41.9 (momoney.com) [1/32/tunnel/down]
127.148.176.10 -> 143.192.152.119 (mbone.dipu.edu)
[1/32/tunnel]
```

For each neighbor of the queried multicast router, the IP of the queried router is displayed, followed by the IP and name of the neighbor. In square brackets the metric (cost of connection), the treashold (multicast ttl) is displayed. If

```
127.148.176.10 -> 105.1.41.9 (momoney.com) [1/32/tunnel/down]
127.148.176.10 -> 143.192.152.119 (mbone.dipu.edu)
```

[1/32/tunnel]

For each neighbor of the queried multicast router, the IP of the queried router is displayed, followed by the IP and name of the neighbor. In square brackets the metric (cost of connection), the treashold (multicast ttl) is displayed. If the queried multicast router has a newer version number, the type (tunnel, srcrt) and status (disabled, down) of the connection is displayed.

#### IMPORTANT NOTE

mrinfo must be run as root.

#### SEE ALSO

mROUTED(8), map-MBONE(8), MTRACE(8)

#### AUTHOR

Van Jacobson

Page 2

## APPENDIX I. *map-mbone* MAN PAGE

MAP-MBONE(8)

UNIX System V

MAP-MBONE(8)

### NAME

*map-mbone* - Multicast connection mapper

### SYNOPSIS

```
/usr/sbin/map-mbone [ -d debug_level ] [ -f ] [ -g ] [ -n ]  
[ -r retry_count ] [ -t timeout_count ] [ starting_router ]
```

### DESCRIPTION

*map-mbone* attempts to display all multicast routers that are reachable from the multicast *starting\_router*. If not specified on the command line, the default multicast *starting\_router* is the localhost.

*map-mbone* traverses neighboring multicast routers by sending the ASK\_NEIGHBORS IGMP message to the multicast *starting\_router*. If this multicast router responds, the version number and a list of their neighboring multicast router addresses is part of that response. If the responding router has recent multicast version number, then *map-mbone* requests additional information such as metrics, thresholds, and flags from the multicast router. For each new occurrence of neighboring multicast router in the reply and provided the flooding option has been selected, then *map-mbone* asks each of this multicast router for a list of neighbors. This search for unique routers will continue until no new neighboring multicast routers are reported.

### INVOCATION

"-d" option sets the debug level. When the debug level is greater than the default value of 0, addition debugging messages are printed. Regardless of the debug level, an error condition, will always write an error message and will cause *map-mbone* to terminate. Non-zero debug levels have level 3

all level 2 messages plus notifications of all packet timeouts are printed to stderr.

"-f" option sets flooding option. Flooding allows the recursive search of neighboring multicast routers and is enable by default when *starting\_router* is not used.

"-g" option sets graphing in GraphEd format.

"-n" option disables the DNS lookup for the multicast routers names.

"-r retry\_count" sets the neighbor query retry limit. Default is 1 retry. "-n" option disables the DNS lookup for the multicast routers names.

"-r retry\_count" sets the neighbor query retry limit. Default is 1 retry.

"-t timeout\_count" sets the number of seconds to wait for a neighbor query reply before retrying. Default timeout is 2 seconds.

#### IMPORTANT NOTE

map-mbone must be run as root.

#### SEE ALSO

mrouted(8), mrinfo(8), mtrace(8)

#### AUTHOR

Pavel Curtis

## APPENDIX J. *mtrace* MAN PAGE

MTRACE(8)

UNIX System V (May 8, 1995)

MTRACE(8)

### NAME

*mtrace* - print multicast path from a source to a receiver

### SYNOPSIS

```
mtrace [ -g gateway ] [ -i if_addr ] [ -l ] [ -M ] [ -m  
max_hops ] [ -n ] [ -p ] [ -q nqueries ] [ -r resp_dest ] [  
-s ] [ -S stat_int ] [ -t ttl ] [ -v ] [ -w waittime ]  
source [ receiver ] [ group ]
```

### DESCRIPTION

Assessing problems in the distribution of IP multicast traffic can be difficult. *mtrace* utilizes a tracing feature implemented in multicast routers (mrouted version 3.3 and later) that is accessed via an extension to the IGMP protocol. A trace query is passed hop-by-hop along the reverse path from the receiver to the source, collecting hop addresses, packet counts, and routing error conditions along the path, and then the response is returned to the requestor.

The only required parameter is the source host name or address. The default receiver is the host running *mtrace*, and the default group is "MBone Audio" (224.2.0.1), which is sufficient if packet loss statistics for a particular multicast group are not needed. These two optional parameters may be specified to test the path to some other receiver in a particular group, subject to some constraints as detailed below. The two parameters can be distinguished because the receiver is a unicast address and the group is a multicast address.

NOTE: For Solaris 2.4/2.5, if the multicast interface is not the default interface, the *-i* option must be used to set the local address.

### OPTIONS

*-g gwy* Send the trace query via unicast directly to the multicast router *gwy* rather than multicasting the query. This must be the last-hop router on the path from the intended source to the receiver.

CAUTION!! Versions 3.3 and 3.5 of mrouted will crash if a trace query is received via a unicast packet and mrouted has no route for the source address. Therefore, do not use the *-g* option unless the target mrouted has been verified to be 3.4 or newer than 3.5.

*-i addr* Use *addr* as the local interface address (on a multi-homed host) for sending the trace query and as the default for the receiver and the response

destination.

- l Loop indefinitely printing packet rate and loss statistics for the multicast path every 10 seconds (see -S stat\_int).
- M Always send the response using multicast rather than attempting unicast first.
- m n Set to n the maximum number of hops that will be traced from the receiver back toward the source. The default is 32 hops (infinity for the DVMRP routing protocol).
- n Print hop addresses numerically rather than symbolically and numerically (saves a nameserver address-to-name lookup for each router found on the path).
- q n Set the maximum number of query attempts for any hop to n. The default is 3.
- p Listen passively for multicast responses from traces initiated by others. This works best when run on a multicast router.
- r host Send the trace response to host rather than to the host on which mtrace is being run, or to a multicast address other than the one registered for this purpose (224.0.1.32).
- s Print a short form output including only the multicast path and not the packet rate and loss statistics.
- S n Change the interval between statistics gathering traces to n seconds (default 10 seconds).
- t ttl Set the ttl (time-to-live, or number of hops) for multicast trace queries and responses. The default is 64, except for local queries to the "all routers" multicast group which use ttl 1.
- v Verbose mode; show hop times on the initial trace and statistics display.
- w n Set the time to wait for a trace response to n seconds (default 3 seconds).

#### USAGE

##### How It Works

The technique used by the traceroute tool to trace unicast



network paths will not work for IP multicast because ICMP responses are specifically forbidden for multicast traffic. Instead, a tracing feature has been built into the multicast routers. This technique has the advantage that additional information about packet rates and losses can be accumulated while the number of packets sent is minimized.

Since multicast uses reverse path forwarding, the trace is run backwards from the receiver to the source. A trace query packet is sent to the last hop multicast router (the leaf router for the desired receiver address). The last hop router builds a trace response packet, fills in a report for its hop, and forwards the trace packet using unicast to the router it believes is the previous hop for packets originating from the specified source. Each router along the path adds its report and forwards the packet. When the trace response packet reaches the first hop router (the router that is directly connected to the source's net), that router sends the completed response to the response destination address specified in the trace query.

If some multicast router along the path does not implement the multicast traceroute feature or if there is some outage, then no response will be returned. To solve this problem, the trace query includes a maximum hop count field to limit the number of hops traced before the response is returned. That allows a partial path to be traced.

The reports inserted by each router contain not only the address of the hop, but also the ttl required to forward and some flags to indicate routing errors, plus counts of the total number of packets on the incoming and outgoing interfaces and those forwarded for the specified group. Taking differences in these counts for two traces separated in time and comparing the output packet counts from one hop with the input packet counts of the next hop allows the calculation of packet rate and packet loss statistics for each hop to isolate congestion problems.

#### Finding the Last-Hop Router

The trace query must be sent to the multicast router which is the last hop on the path from the source to the receiver. If the receiver is on the local subnet (as determined using the subnet mask), then the default method is to multicast the trace query to all-routers.mcast.net (224.0.0.2) with a ttl of 1. Otherwise, the trace query is multicast to the group address since the last hop router will be a member of that group if the receiver is. Therefore it is necessary to specify a group that the intended receiver has joined. This multicast is sent with a default ttl of 64, which may not be sufficient for all cases (changed with the -t option). If the last hop router is known, it may also be addressed

directly using the -g option). Alternatively, if it is desired to trace a group that the receiver has not joined, but it is known that the last-hop router is a member of another group, the -g option may also be used to specify a different multicast address for the trace query.

When tracing from a multihomed host or router, the default receiver address may not be the desired interface for the path from the source. In that case, the desired interface should be specified explicitly as the receiver.

#### Directing the Response

By default, mtrace first attempts to trace the full reverse path, unless the number of hops to trace is explicitly set with the -m option. If there is no response within a 3 second timeout interval (changed with the -w option), a "\*" is printed and the probing switches to hop-by-hop mode. Trace queries are issued starting with a maximum hop count of one and increasing by one until the full path is traced or no response is received. At each hop, multiple probes are sent (default is three, changed with -q option). The first half of the attempts (default is one) are made with the unicast address of the host running mtrace as the destination for the response. Since the unicast route may be blocked, the remainder of attempts request that the response be multicast to mtrace.mcast.net (224.0.1.32) with the ttl set to 32 more than what's needed to pass the thresholds seen so far along the path to the receiver. For the last quarter of the attempts (default is one), the ttl is increased by another 32 each time up to a maximum of 192. Alternatively, the ttl may be set explicitly with the -t option and/or the initial unicast attempts can be forced to use multicast instead with the -M option. For each attempt, if no response is received within the timeout, a "\*" is printed. After the specified number of attempts have failed, mtrace will try to query the next hop router with a DVMRP\_ASK\_NEIGHBORS2 request (as used by the mrinfo program) to see what kind of router it is.

#### EXAMPLES

The output of mtrace is in two sections. The first section is a short listing of the hops in the order they are queried, that is, in the reverse of the order from the source to the receiver. For each hop, a line is printed showing the hop number (counted negatively to indicate that this is the reverse path); the multicast routing protocol (DVMRP, MOSPF, PIM, etc.); the threshold required to forward data (to the previous hop in the listing as indicated by the up-arrow character); and the cumulative delay for the query to reach that hop (valid only if the clocks are synchronized). This first section ends with a line showing the round-trip time which measures the interval from when

the query is issued until the response is received, both derived from the local system clock. A sample use and output might be:

```
oak.isi.edu 80# mtrace -l caraway.lcs.mit.edu 224.2.0.3
Mtrace from 18.26.0.170 to 128.9.160.100 via group 224.2.0.3
Querying full reverse path...
 0 oak.isi.edu (128.9.160.100)
-1 cub.isi.edu (128.9.160.153)  DVMRP  thresh^ 1  3 ms
-2 la.dart.net (140.173.128.1)  DVMRP  thresh^ 1 14 ms
-3 dc.dart.net (140.173.64.1)   DVMRP  thresh^ 1 50 ms
-4 bbn.dart.net (140.173.32.1)  DVMRP  thresh^ 1 63 ms
-5 mit.dart.net (140.173.48.2)  DVMRP  thresh^ 1 71 ms
-6 caraway.lcs.mit.edu (18.26.0.170)
Round trip time 124 ms
```

The second section provides a pictorial view of the path in the forward direction with data flow indicated by arrows pointing downward and the query path indicated by arrows pointing upward. For each hop, both the entry and exit addresses of the router are shown if different, along with the initial ttl required on the packet in order to be forwarded at this hop and the propagation delay across the hop assuming that the routers at both ends have synchronized clocks. The right half of this section is composed of several columns of statistics in two groups. Within each group, the columns are the number of packets lost, the number of packets sent, the percentage lost, and the average packet rate at each hop. These statistics are calculated from differences between traces and from hop to hop as explained above. The first group shows the statistics for all traffic flowing out the interface at one hop and in the interface at the next hop. The second group shows the statistics only for traffic forwarded from the specified source to the specified group.

These statistics are shown on one or two lines for each hop. Without any options, this second section of the output is printed only once, approximately 10 seconds after the initial trace. One line is shown for each hop showing the statistics over that 10-second period. If the -l option is given, the second section is repeated every 10 seconds and two lines are shown for each hop. The first line shows the statistics for the last 10 seconds, and the second line shows the cumulative statistics over the period since the initial trace, which is 101 seconds in the example below. The second section of the output is omitted if the -s option is set.

Waiting to accumulate statistics... Results after 101 seconds:

```
Source Response Dest  Packet Statistics For Only For Traffic
```

```

18.26.0.170 128.9.160.100 All Multicast Traffic From 18.26.0.170
| / rtt 125 ms Lost/Sent = Pct Rate To 224.2.0.3
v / hop 65 ms -----
18.26.0.144
140.173.48.2 mit.dart.net
| ^ ttl 1 0/6 = --% 0 pps 0/2 = --% 0 pps
v | hop 8 ms 1/52 = 2% 0 pps 0/18 = 0% 0 pps
140.173.48.1
140.173.32.1 bbn.dart.net
| ^ ttl 2 0/6 = --% 0 pps 0/2 = --% 0 pps
v | hop 12 ms 1/52 = 2% 0 pps 0/18 = 0% 0 pps
140.173.32.2
140.173.64.1 dc.dart.net
| ^ ttl 3 0/271 = 0% 27 pps 0/2 = --% 0 pps
v | hop 34 ms -1/2652 = 0% 26 pps 0/18 = 0% 0 pps
140.173.64.2
140.173.128.1 la.dart.net
| ^ ttl 4 -2/831 = 0% 83 pps 0/2 = --% 0 pps
v | hop 11 ms -3/8072 = 0% 79 pps 0/18 = 0% 0 pps
140.173.128.2
128.9.160.153 cub.isi.edu
| ^ ttl 5 833 83 pps 2 0 pps
v | hop -8 ms 8075 79 pps 18 0 pps
128.9.160.100 128.9.160.100
Receiver Query Source

```

Because the packet counts may be changing as the trace query is propagating, there may be small errors (off by 1 or 2) in these statistics. However, those errors should not accumulate, so the cumulative statistics line should increase in accuracy as a new trace is run every 10 seconds. There are two sources of larger errors, both of which show up as negative losses:

- o If the input to a node is from a multi-access network with more than one other node attached, then the input count will be (close to) the sum of the output counts from all the attached nodes, but the output count from the previous hop on the traced path will be only part of that. Hence the output count minus the input count will be negative.
- o In release 3.3 of the DVMRP multicast forwarding software for SunOS and other systems, a multicast packet generated on a router will be counted as having come in an interface even though it did not. This creates the negative loss that can be seen in the example above.

Note that these negative losses may mask positive losses.

In the example, there is also one negative hop time. This simply indicates a lack of synchronization between the

system clocks across that hop. This example also illustrates how the percentage loss is shown as two dashes when the number of packets sent is less than 10 because the percentage would not be statistically valid.

A second example shows a trace to a receiver that is not local; the query is sent to the last-hop router with the -g option. In this example, the trace of the full reverse path resulted in no response because there was a node running an old version of mrouted that did not implement the multicast traceroute function, so mtrace switched to hop-by-hop mode. The Route pruned error code indicates that traffic for group 224.2.143.24 would not be forwarded.

```
oak.isi.edu 108# mtrace -g 140.173.48.2 204.62.246.73 \
                                butter.lcs.mit.edu 224.2.143.24
Mtrace from 204.62.246.73 to 18.26.0.151 via group
224.2.143.24
Querying full reverse path... * switching to hop-by-hop:
0  butter.lcs.mit.edu (18.26.0.151)
-1  jam.lcs.mit.edu (18.26.0.144)  DVMRP  thresh^ 1  33 ms
    Route pruned
-2  bbn.dart.net (140.173.48.1)  DVMRP  thresh^ 1  36 ms
-3  dc.dart.net (140.173.32.2)  DVMRP  thresh^ 1  44 ms
-4  darpa.dart.net (140.173.240.2)  DVMRP  thresh^ 16  47 ms
-5  * * * noc.hpc.org (192.187.8.2) [mrouted 2.2] didn't
    respond
Round trip time 95 ms
```

#### AUTHOR

Implemented by Steve Casner based on an initial prototype written by Ajit Thyagarajan. The multicast traceroute mechanism was designed by Van Jacobson with help from Steve Casner, Steve Deering, Dino Farinacci, and Deb Agrawal; it was implemented in mrouted by Ajit Thyagarajan and Bill Fenner. The option syntax and the output format of mtrace are modeled after the unicast traceroute program written by Van Jacobson.

#### SEE ALSO

mrouted(8), mrinfo(8), map-mbone(8), traceroute(8)



## APPENDIX K. *crontab* MAN PAGE (UNIX)

CRONTAB(1)

CRONTAB(1)

### NAME

crontab - user crontab file

### SYNOPSIS

```
crontab [file]
crontab -r
crontab -l
```

### DESCRIPTION

crontab copies the specified file, or standard input if no file is specified, into a directory that holds all users' crontabs. The -r option removes a user's crontab from the crontab directory. crontab -l will list the crontab file for the invoking user.

Users are permitted to use crontab if their names appear in the file /usr/lib/cron/cron.allow. If that file does not exist, the file /usr/lib/cron/cron.deny is checked to determine if the user should be denied access to crontab. If neither file exists, only root is allowed to submit a job. If cron.allow does not exist and cron.deny exists but is empty, global usage is permitted. The allow/deny files consist of one user name per line.

A crontab file consists of lines of six fields each. The fields are separated by spaces or tabs. The first five are integer patterns that specify the following:

- minute (0-59),
- hour (0-23),
- day of the month (1-31),
- month of the year (1-12),
- day of the week (0-6 with 0=Sunday).

Each of these patterns may be either an asterisk (meaning all legal values) or a list of elements separated by commas. An element is either a number or two numbers separated by a minus sign (meaning an inclusive range). Note that the specification of days may be made by two fields (day of the month and day of the week). If both are specified as a list of elements, both are adhered to. For example, 0 0 1,15 \* 1 would run a command on the first and fifteenth of each month, as well as on every Monday. To specify days by only one field, the other field should be set to \* (for example, 0 0 \* \* 1 would run a command only on Mondays).

The sixth field of a line in a crontab file is a string that is executed by the shell at the specified times. A percent character in this field (unless escaped by \) is translated to a new-line character. Only the first line (up to a % or end of line) of the command field is executed by the shell. The other lines are made available to the command as standard input.

The shell is invoked from your \$HOME directory with an arg0 of sh. Users who desire to have their .profile executed must explicitly do so in the crontab file. Cron supplies a default environment for every shell,

Page 1

defining HOME, LOGNAME, USER, SHELL(=/bin/sh),  
PATH(=/usr/sbin:/usr/bsd:/usr/bin:/bin:/etc:/usr/etc), and TZ.

If you do not redirect the standard output and standard error of your  
commands, any generated output or errors will be mailed to you.

Any errors encountered while parsing the crontab file (or stdin) will  
cause crontab to abort with no changes being made to any existing  
crontab.

#### FILES

/usr/lib/cron	main cron directory
/usr/spool/cron/crontabs	spool area
/usr/lib/cron/log	accounting information
/usr/lib/cron/cron.allow	list of allowed users
/usr/lib/cron/cron.deny	list of denied users

#### SEE ALSO

sh(1), cron(1M)

#### WARNINGS

If you inadvertently enter the crontab command with no argument(s), you  
may exit without overwriting the existing crontab entry either by  
generating an interrupt (typically by typing ^C or DEL), or by providing  
empty input (typically by typing ^D on the first empty line).

If a job is scheduled during the 'witching hour' - the time during a  
change from the main to alternate time zone, the job will either be run  
once (if the actual time exists twice) or not at all (if the actual time  
never exists).



## APPENDIX L. *rshstl* EXECUTABLE

```
#!/usr/bin/perl

# Program:      rshstl
#
# Purpose:      rshstl runs a user-specified command against all STL
#               hosts
#
# Example:      rshstl produces a 'who' listing
#               executed remotely on each host
#
# Author:       Don Brutzman
#
# Revised:      27 June 96
#
# URL:          http://www.stl.nps.navy.mil/~brutzman/perl/rshstl
#
# Reference:    Wall, Larry and Schwartz, Randal L., _Programming
#               perl_, O'Reilly & Associates, Sebastopol California,
#               1991.
#
# Bugs:         The error message 'rshstl: Command not found.' likely
#               indicates that perl was not available or not in
#               /usr/bin, rather than 'rshstl' itself not being
#               visible. Try 'which perl' on the host which failed to
#               find out.

@hosts = (
    "azure.stl.nps.navy.mil",
    "baby.stl.nps.navy.mil",
    "blackand.stl.nps.navy.mil",
    "cadet.stl.nps.navy.mil",
    "cirrus.stl.nps.navy.mil",
    "fletch.stl.nps.navy.mil",
    "navy.stl.nps.navy.mil",
    "royal.stl.nps.navy.mil",
    "slate.stl.nps.navy.mil",
    "steel.stl.nps.navy.mil",
);

if (!@ARGV) {
    print STDERR ("Usage:  rshstl _your_command_here_\n");
    exit;
}
foreach $host (@hosts) {
    print STDERR "\nrsh $host @ARGV\n";
    system      ("rsh $host @ARGV");
}
exit;
```



## APPENDIX M. *ping\_host.pl* EXECUTABLE

```
#!/usr/local/bin/perl

# Program:      ping_host.pl
#
# Purpose:      verify status of network hosts using ping
#
# Author:       Evan Edwards
#
# Revised:      20 June 96
#
# URL:          http://www.stl.nps.navy.mil/
#               ~iirg/atm/monitoring/Ping_pages/NPS
#
# Reference:    Wall, Larry and Schwartz, Randal L., _Programming
#               perl_, O'Reilly & Associates, Sebastopol California,
#               1991.
#
# Bugs:         stalled execution usually means a mismatch between the
#               version of ping in use by the o/s and the program

#####
#location dependant variables

$local_name = "NPS";
$local_URL  = "http://www.stl.nps.navy.mil/
~iirg/atm/monitoring/Ping_pages/NPS";
$baynet     = "http://www.stl.nps.navy.mil/
~iirg/atm/monitoring/Ping_pages/baynet_monitoring.html";
$email_notification = "edwardse";
$refresh_rate = 60;
#####

#####
#Subroutine host_find
#looks in the same directory as the executable for host files
#host file contain host names, comments or black lines
#chops the last part of the name
#stores it in @domains which is used to iteratively loop through the
#main program

sub host_find {

    open (HOSTS, "ls hosts.txt.* |") || die("could not open host file");
    @hosts = <HOSTS>;
    close (HOSTS);

    foreach $domain (@hosts) {
        @dom = split (/\/\./,$domain);
        $dom = pop(@dom);
        push(@domains, $dom);
    }
    return;
}#host_find
#####
```

```
#####
#Subroutine date_split
#put the date into an array and split it
#gets the mon, day and yr

sub date_split {

    open (DATE, "date\n |") || die("date not available");

    $date = (<DATE>);
    @date= split (/s+/, $date);
    $year = pop @date;
    $zone = pop @date;
    $time = pop @date;
    $calday = pop @date;
    $month = pop @date;
    $day = pop @date;
    @time = split (/:/, $time);
    $time = $time[0].$time[1];
    return;

}&date_split;
#
#####

#MAIN

open (FILEOUT, ">status_old.html") || die("could not open status.html");

print FILEOUT <<ENDOT;
<html>
<head>
<title>$local_name Monitoring Page</title>
</head>
<body bgcolor="#ffffff" text="#000000" link="#0000ee" vlink="551a8b"
alink="ff0000">

<META HTTP-EQUIV="Refresh" CONTENT=$refresh_rate>

<h1>$local_name Monitoring Page</h1>
<hr>
<h3>Ping from $local_name to various BayNet sites</h3>
ENDOT

&date_split;

print FILEOUT ("<h3> $date </h3>");

print FILEOUT  ("\n<PRE>\n");

&host_find;

foreach $domain (@domains) {
    print FILEOUT ("<HR><strong>$domain<BR></strong><BR><A
HREF=\"$local_URL/hosts.txt.$domain\">hosts tested</A><BR><HR>\n");

    open (HOSTS, "hosts.txt.$domain") || die("could not find
hosts.txt.$domain");

    while ($host_line = <HOSTS>) {
```

```

chop $host_line;

if (!( $host_line =~ /^[#]/ )) { # if the first character is not "#"
    if (!( $host_line =~ /^[" "]/ )) { # first character not a space
        if (!( $host_line =~ /^$/ )) { # if not empty line

($host) = split (/\\s/, $host_line);
# remainder of line ignored, assign list to list

# DOWN Status Check
if (-e "$host.down") {

# DOWN Status exists #####
    open (PING, "rsh azure ping $host|" ) || die ("could not ping");

    @ping = ();

    foreach $pingline (<PING>) {
        @ping = (@ping, $pingline);
    }

# Good ping
    open (DOWNFILE, "$host.down") || die "could not open $host.down" ;
    if ( ($ping[0] =~ /alive/) || ($ping[1] =~ /time=/) ) {
        # Sun || SGI

        print (FILEOUT '<FONT color="green">');
        print (FILEOUT "<strong>$host is alive</strong><BR>");
        print (FILEOUT '<FONT color="black">');

        open (UPFILE, ">Up/$host.$year.$month.$day.$time") || die
("could not open $host.$year.$month.$day.$time");
        print UPFILE ("\n$host is alive at $date\n");
        print UPFILE ("Previous unresponsive pings:");
        print UPFILE (<DOWNFILE> );
        system ("rm $host.down");

        close (UPFILE);
        close (PING);
        close (DOWNFILE);

        if ($email_notification) {
            open (UPMSG, "|mail $email_notification <
Up/$host.$year.$month.$day.$time\n");
            close (UPMSG);
        }
    } #if

# Bad ping
    else {
        open (DOWNFILE, ">>$host.down") || die "could not open
$host.down" ;
        print DOWNFILE (" $host was dead at $date");
        close (DOWNFILE);
        print (FILEOUT '<FONT color="red">');
        print (FILEOUT '<blink><BR>');
        print (FILEOUT "<strong>$host is dead!</strong><BR>");
        print (FILEOUT '</blink><BR>');
        print (FILEOUT '<FONT color="black">');
    } #else

```

```

    }# end of DOWN Status exists

    #DOWN Status does NOT exist
    #####
    else {

        open (PING,"rsh azure ping $host|") || die("could not ping");

        @ping = ();

        foreach $pingline (<PING>) {
            @ping = (@ping, $pingline);
        }

        if ( ($ping[0] =~ /alive/) || ($ping[1] =~ /time=/) ) {
            # Sun || SGI

            # good ping #####
            print (FILEOUT '<FONT color="green">');
            print FILEOUT ( "$host is alive\n");
            print (FILEOUT '<FONT color="black">');
        }# if
        # bad ping #####
        else {
            open (DOWNFILE, ">$host.down") || die("cannot open downfile
                for $host");
            print DOWNFILE ("\n$host was dead at $date");
            close (DOWNFILE);

            open (DOWNMSG, "|mail $email_notification\n");
            print DOWNMSG ("Monitoring has detected that $host was dead
                at $date\n\n");
            print DOWNMSG ("full results at $local_URL/$host.down\n");
            close (DOWNMSG);

            print (FILEOUT '<FONT color="red">');
            print (FILEOUT '<blink>');
            print (FILEOUT "<strong>$host is dead!</strong>");
            print (FILEOUT '</blink>');
            print (FILEOUT '<FONT color="black">');

        }#else

        close (PING);

    }#else - end of DOWN status does not exist section

        }## if nothing is on the line ## blank lines are ignored
        else {
        }
        }## if the first character is a space
        else {
            print FILEOUT (" $host_line\n");
        }
        }## if the first character is a "#"
        else {
            print FILEOUT (" $host_line\n");
        }
    }#while
}#foreach

```

```
print FILEOUT <<EOT2;  
</PRE>  
<HR>  
<A HREF= $baynet> | Baynet Monitoring Page |</A><BR>  
<A HREF= "monitoring.html" > | $local_name Monitoring Page | </A>  
</body>  
</html>  
EOT2
```

```
close (FILEOUT);  
system ("cp status_old.html status.html")
```





## APPENDIX N. *int\_ping.cgi* EXECUTABLE

```
#!/usr/local/bin/perl

# Program:      int_ping.cgi
#
# Purpose:      interactively verify status of network hosts with ping
#
# Author:       Evan Edwards
#
# Revised:      20 June 96
#
# URL:          http://www.stl.nps.navy.mil/
#               ~iirg/atm/monitoring/Ping_pages/NPS/interactive/
#
# Reference:    Brenner, Steven & Aoki, Edwin, _Introduction to
#               CGI/PERL_, M&T Books, New York, NY, 1996.
#
# Bugs:         currently must be run from blackand due to cgi setup

require "cgi-lib.pl";

$server = 'blackand.stl.nps.navy.mil';
$hostnamefile = "hosts.txt.stl";

#####
#
#Program:      Subroutine host_find
#Author:       Evan Edwards
#Date:         21 May 1996
#Purpose:      looks in current directory host files
#               chops the last part of the name &
#               stores it in @domains which is used by main

sub host_find {

    open (HOSTS, "ls hosts.txt.* |") || die("could not find host");
    @hosts = <HOSTS>;
    close (HOSTS);

    foreach $domain (@hosts) {
        @dom = split (/\.\/,$domain);
        $dom = pop(@dom);
        push(@domains, $dom);
    }
    return;
}#host_find
#
#####
```

```
#####
#
#Program:    Subroutine BuildHostMenu (launcher.cgi embellishment
#Author:     Don Brutzman
#Date:       21 May 1996
#

sub BuildHostMenu {

    $x=0;
    $selectvarname = $_[0];
    &host_find;

    printf ("<UL>\n");

    foreach $network (@domains) {

        if (!<HOSTNAMEFILE>) {
            open (HOSTNAMEFILE, "hosts.txt.$network")
                || die "cannot open $network: $!";
        }

        printf ("<HR><Li>  <H2>$network</H2>\n");
        # printf ("<Li>  <STRONG>$selectvarname</STRONG><BR>\n");

        foreach $lhost (<HOSTNAMEFILE>) {
            chop $lhost;

            if ( (!($lhost =~ /^[#]/)) && (!($lhost =~ /^[ " ]/))
                && (!($lhost =~ /^$/)) ) {
                # if first char not "#" OR not a space OR not an empty line

                printf ("<input type=checkbox name=phost value=\"$lhost\">
                    $lhost\n<BR>");

                }#if
        }#foreach
    }
    printf ("</UL>\n");

    return;
}
#
#####
```

MAIN:

```
{
  if (&ReadParse(*input)){
    &ProcessForm;
  }
  else {
    &ShowForm;
  }
}
```

#####

```
#
#Program:    Subroutine showform
#Author:     Evan Edwards
#Date:       21 May 1996
#Purpose:    HTML form that is interface to int_ping.cgi
```

sub ShowForm

```
{
  print &PrintHeader;
  print &HtmlTop("Ping CGI");
  print <<EOT;
url for this page is:<BR>
http://$server/~iirg/atm/monitoring/Ping_pages/NPS/interactive/interacti
ve.cgi<BR>
<HR>
<H2>Other Tools </H2>
<UL>
  <Li>  <H3><I>traceroute</I> is <A HREF=
        \"traceroute/traceroute.cgi\"> traceroute </A></H3>

  <Li>  <H3><I>nslookup</I> is <A HREF= \"nslookup/nslookup.cgi\">
        nslookup </A></H3>

</UL>

<HR>

<H2>Choose one or more of these hosts to ping</H2>
<H4>The test will be performed from <I>$server</I></H4>

<form method=POST>
EOT

&BuildHostMenu("hosts to ping");

print <<EOT2;
<input type=submit value="ping host">
</form>
EOT2

  print &HtmlBot;
}
```

#####

```
#####
#
#Program:   Subroutine ProcessForm (embelished for int_ping.cgi)
#Authors:   Don Brutzman & Evan Edwards
#Date:      21 May 1996

sub ProcessForm
{
    print &PrintHeader;
    print &HtmlTop( "Interactive Network Monitoring from $server");

    open (DATE, "date\n |") || die("date not available");
    print ("<H3>", <DATE>, "</H3><HR>");
    close (DATE);

    @phosts = split("\0", $input{'phost'});
    #puts multiple select into array
    foreach $lhost (@phosts) {
        open (PING, "ping -c 1 $lhost|") || die("could not ping $lhost");
        print ("<H2> Ping to $lhost </H2>");
        print ("Command Line equivalent: <I>ping -c 1 $lhost</I><BR>");
        print ("<PRE>");
        print (<PING>, "</PRE><BR><HR>");
        close (PING);
    }

    @thosts = split("\0", $input{'thost'});
    foreach $lhost (@thosts) {
        open (TR, "traceroute $lhost|")
            || die("could not traceroute $lhost");
        print ("<H2> Traceroute to $lhost </H2>");
        print ("Command Line equivalent:<I>traceroute $lhost</I><BR>");
        print ("<PRE>");
        print (<TR>, "</PRE><BR><HR>");

        close (TR);
    }

    print ("<A HREF= \"interactive.cgi\"> | ping | </A>");
    print ("<A HREF= \"traceroute/traceroute.cgi \">traceroute | </A>");
    print ("<A HREF= \"nslookup/nslookup.cgi\"> nslookup | </A>");
    print &HtmlBot;
}
#
#####
```

## APPENDIX O. *traceroute.cgi* EXECUTABLE

```
#!/usr/local/bin/perl

# Program:      traceroute.cgi
#
# Purpose:      interactively traceroute network hosts with ping
#
# Author:       Evan Edwards
#
# Revised:      20 June 96
#
# URL:          http://www.stl.nps.navy.mil/~iirg/atm/
#               monitoring/Ping_pages/NPS/interactive/traceroute/
#
# Reference:    Brenner, Steven & Aoki, Edwin, _Introduction to
#               CGI/PERL_, M&T Books, New York, NY, 1996.
#
# Bugs:         currently must be run from blackand due to cgi setup

require "cgi-lib.pl";

$server = 'blackand.stl.nps.navy.mil';
$hostnamefile = "hosts.txt";

#####
#
#Program:      Subroutine host_find
#Author:       Evan Edwards
#Date:         21 May 1996
#Purpose:      looks in current directory host files
#               chops the last part of the name &
#               stores it in @domains which is used by main

sub host_find {

    open (HOSTS, "ls hosts.txt.* |") || die("could not find host");
    @hosts = <HOSTS>;
    close (HOSTS);

    foreach $domain (@hosts) {
        @dom = split (/\.\/,$domain);
        $dom = pop(@dom);
        push(@domains, $dom);
    }
    return;
}#host_find
#
#####
```

```
#####
#
#Program:    Subroutine BuildHostMenu (launcher.cgi embellishment)
#Authors:    Don Brutzman & Evan Edwards
#Date:       21 May 1996
#

sub BuildHostMenu {

$selectvarname = $_[0];

foreach $network (@domains) {

    open (HOSTNAMEFILE, "trace.$network") || die "cannot open $network: $!";
    @HOSTS = <HOSTNAMEFILE>;

    print ("<TR>\n");
    print ("<th align=center> $network\n");
    printf ("<th align=left><select name=thost size=2 multiple>");

    foreach $lhost (@HOSTS) {

        if((!($lhost=~ /^[#]/))&&(!($lhost=~ /^[ "]/))&&(!($lhost=~ /^$/))) {
            # if first char not "#" OR not a space OR if not an empty line
            printf ("<option> $lhost\n");
        }

    }#foreach
    printf ("</select>\n<BR>");

}

return;
}

#####
```

MAIN:

```
{
  if (&ReadParse(*input)){
    &ProcessForm;
  }
  else {
    &ShowForm;
  }
}
```

#####  
#

#Program: Subroutine showform  
#Author: Evan Edwards  
#Date: 21 May 1996  
#Purpose: HTML form that is interface to traceroute.cgi

sub ShowForm  
{

print &PrintHeader;  
print &HtmlTop("traceroute CGI");  
  
print <<EOT;  
url for this page is:<BR>  
http://\$server/~iirg/atm/monitoring/Ping\_pages/NPS/  
interactive/traceroute/traceroute.cgi<BR>  
<HR>  
<P><H2>traceroute one or more hosts you select.</H2>  
<H4>The test will be performed from <I>\$server</I></H4>

<form method=get>  
<table border cellpadding=5 cellspacing=15>  
<TR>  
    <th align=left> <H3>network</H3>  
    <th align=left> <H3>hosts</H3>

EOT

&host\_find;

&BuildHostMenu("hosts to trace");

print <<EOT2;  
</table>  
<input type=submit value="trace">  
</form>  
EOT2

print &HtmlBot;  
}

#####

```
#####
#
#Program:   Subroutine ProcessForm (embelished for traceroute.cgi)
#Author:    Don Brutzman & Evan Edwards
#Date:      21 May 1996

sub ProcessForm
{
print &PrintHeader;
print &HtmlTop( "traceroute from $server");

open (DATE, "date\n |") || die("date not available");
print ("<H3>", <DATE>, "</H3><HR>");
close (DATE);

@thosts = split("\0", $input{'thost'});
foreach $lhost (@thosts) {
    open (TR, "traceroute $lhost|") || die("could not trace $lhost");

    print ("<H2> Traceroute to $lhost </H2>");
    print ("Command Line equivalent: <I>traceroute $lhost</I><BR>");
    print ("<PRE><BR>");
    print (<TR>, "\n");
    print ("</PRE><HR>");

    close (TR);
}

print ("<A HREF= \"../interactive.cgi\"> | ping | </A><BR>");
print ("<A HREF= \"traceroute.cgi\" > |traceroute | </A><BR>");
print ("<A HREF= \"../nslookup/nslookup.cgi\" >|nslookup|</A><BR>");
print &HtmlBot;
}

#####
[DHRSC1]
```



## APPENDIX P. *nslookup.cgi* EXECUTABLE

```
#!/usr/local/bin/perl

# Program:      nslookup.cgi
#
# Purpose:      interactively nslookup network hosts
#
# Author:       Evan Edwards
#
# Revised:      21 May 96
#
# URL:          http://www.stl.nps.navy.mil/~iirg/atm/
#               monitoring/Ping_pages/NPS/interactive/nslookup/
#
# Reference:    Brenner, Steven & Aoki, Edwin, _Introduction to
#               CGI/PERL_, M&T Books, New York, NY, 1996.
#
# Bugs:         currently must be run from blackand due to cgi setup
```

```
require "cgi-lib.pl";
```

```
$server = 'blackand.stl.nps.navy.mil';
$hostnamefile = "hosts.txt";
```

```
MAIN:
```

```
{
    if (&ReadParse(*input)){
        &ProcessForm;
    }
    else {
        &ShowForm;
    }
}
```

```
#####
#
#Program:   Subroutine showform
#Author:    Evan Edwards
#Date:      21 May 1996
#Purpose:   HTML form that is interface to nslookup.cgi

sub ShowForm
{
    print &PrintHeader;
    print ("<HR><BR>");
    print &HtmlTop($filename);
    print <<EOT;
    url is
    http://$server/~iirg/atm/monitoring/Ping_pages/NPS/interactive/nslookup/
    $filename<BR>
    <HR>
    <P><H2> This is an interactive program to query Internet domain name
    servers.
    <BR>
    The test will be performed from <I>$server</I></H2>
    <HR>

    <form method=POST>
    <H2>Choose a host - server info will be returned</H2>
    <UL>
        <Li> <input type=checkbox name=nhost value="stl.nps.navy.mil">
        <STRONG>System Technology Lab</STRONG>
        <Li> <input type=checkbox name=nhost value="cs.nps.navy.mil">
        <STRONG>Computer Science</STRONG>
        <Li> <input type=checkbox name=nhost value="nps.navy.mil">
        <STRONG>Computer Center</STRONG><BR>
    </UL>

    <HR>

    <input type=submit value="lookup hosts">
    </form>
    EOT

    print &HtmlBot;
}

#####
```

```
#####
#
#Program:   Subroutine ProcessForm (embelished for nslookup.cgi)
#Author:    Don Brutzman & Evan Edwards
#Date:      21 May 1996

sub ProcessForm
{
print ("content-type: text/html\n\n");
print ("<html>\n");
print ("<head>\n");
print ("<title> nslook from $server </title>\n");
print ("</head>\n");
print ("<body>\n");
print ("<H2> Interactive version nslookup </H2>");

open (DATE, "date\n |") || die("date not available");
print ("<H3>", <DATE>, "</H3><HR>");
close (DATE);

@nhosts = split("\0", $input{'nhost'});
#puts multiple select into array
foreach $lhost (@nhosts) {

    print ("nameserver is <I>$lhost</I><BR>");
    print ("<PRE>");

    open (NS1, "nslookup - $lhost < nslookup.txt.$lhost|");
    print (<NS1>);
    print ("</PRE><BR><HR>");
    close (NS1);

}

print ("<A HREF= \"../interactive.cgi\"> |Interactive Test|</A>\n");
print ("<A HREF= $filename> nslookup | </A>\n");
print ("<A HREF= \"../traceroute/traceroute.cgi\"> traceroute|
</A>\n<BR>");
print ("<A HREF= \"../../monitoring.html\"> | NPS Monitoring Page
|</A>\n");
print ("<A HREF= \"../../../baynet_monitoring.html\"> BayNet
Monitoring Page | </A>\n");
print &HtmlBot;
}

#####
[DHRSC1]
```



## APPENDIX Q. INSTALLATION PROCEDURES

### 1. DIRECTORY AND FILE CREATION

Decide where the monitoring directory will be and who will own the account.

Decide which hosts need to be tested and from where. Create the following directory structure

```
/monitoring  
/monitoring/UP
```

Next configure the host files with the testable hosts. Start with a small selection of hosts for ease of troubleshooting. The host files may be configured in many different ways. The manner in which the information in them is handled depends on the `ping_hosts.pl` code. Blank lines, comments and other information on the host lines can be ignored or reproduced to output. If it is mistakenly passed to the `ping` command the program execution will fail.

Initially, put only host names or IP addresses in the file, one per line as follows:

```
% more hosts.txt.stl  
steel.stl.nps.navy.mil  
navy.stl.nps.navy.mil  
cadet.stl.nps.navy.mil
```

Put all of the files in the same directory. Set the permissions as follows:

```
chmod 755 ping_hosts.pl  
chmod 755 hosts.txt.*
```

### 2. SYSTEM VERIFICATION

#### **a. *ping* format and syntax**

At the command line, enter `ping`. Determine if the count needs to be set by using a switch and verify the output format, either verbose or terse. Resolve the exact syntax and from which host *ping* will be configured for `ping_hosts.pl`. This is covered in detail below.

#### **b. *perl***

At the command line type `perl -v` to determine the version of *perl* that is installed. whereis `perl` can be used to check the path to the *perl* directory. It is vital to execution that the path be set correctly in the first line of `ping_hosts.pl`.

#### **c. date format**

Type `date` at the command line. The format specified in the code contains six fields ordered:

- day of the week
- month
- date
- time
- timezone
- year

Output from `date` varies across platforms in which case subroutine `sub date_find` must be altered.

### **3. PORTING `ping_hosts.pl`**

#### **a. local variable**

Change the \$local\_name to the appropriate title (NPS, MLML, UCSC etc).

Change the \$local\_URL to the full path to the monitoring directory

<http://www.nps.navy.mil/~iirg/atm/monitoring> (do not end with a period or a slash).

Change the e-mail address to the appropriate recipient.

## **b. PING**

The stream PING is opened in two places in the code. PING opens the command for the *ping* of the hosts. In each line the code must be changed as per system configuration and personal desire.

### **1. rsh**

The *ping* command may be used in conjunction with an rshell to another host. This may be desired for a variety of reasons. The clock daemon may run differently on different hosts. *ping* may respond different as well. The particular output of one host may be preferred over another.

```
open (PING, `rsh azure ping $host|`)
```

### **2. ping path**

If *ping* is not resident where the crontab script is run the path to *ping* may need to be specified. The location of *ping* is not consistent on all systems.

```
open (PING, `/usr/etc/ ping $host|`)
```

Change the \$local\_URL to the full path to the monitoring directory

<http://www.nps.navy.mil/~iirg/atm/monitoring> (do not end with a period or slash)

### **3. switches**

*ping* is configured differently depending on the operating system. Once the input and output configuration is determined the switches can be set correctly in the script. For example, if *ping* is configured to stop only with ^C then the number of *pings* must be specified as in one of the two examples below.

```
open (PING, 'ping -c 1 $host|')  
open (PING, 'ping 3 $host|')
```

## **C. EXECUTION**

### **1. rights**

The rights permission must be set correctly on `ping_hosts.pl` as follows:

```
chmod 755 ping_hosts.pl
```

The hosts files need only have read permission.

### **2. host\_find.pl**

The subroutine `host_find.pl` looks through the directory in which `ping_hosts.pl` is executed for host files of the form `hosts.txt.dom`, where `dom` is the domain of the hosts such as NPS, MLML or UCSC.

All the domains that are represented by `host.txt.dom` files are loaded into an array which is then made available for testing in the main program. `host_find.pl` is called once during `ping_hosts.pl`.

### **3. ping\_host.pl execution**



Initially run the command `perl -w ping_hosts.pl` from the command line with a host file that is small and contains only local hosts. `perl -w` provides extra debugging information. The program will execute quickly except in the case of a hung *ping*.

The program creates the file `status_old.html` by opening it for writing. It then sets up the HTML page with the appropriate heading, title and date. The domain for testing is provided by the array that is filled by `host_find.pl`. Next the hosts are taken from the domain text file. One by one they are tested. The testing algorithm is listed below.

*a. down status exists*

Down status is formulated by the existence of a down status file. The file is created on the first negative *ping* result. The full host name is concatenated with 'down' and save to the same directory as the executable file. The down files contain all information pertaining to the down period of the host.

```
% ls
Up/
colum.cse.ucsc.edu.down
hosts.txt.mlml
hosts.txt.nps
hosts.txt.ucsc
ping_hosts.pl
status.html
status_old.html
steel.stl.nps.navy.mil.down
```

In this case, two down status files exist. When down hosts are tested, two actions can occur. In the event of a good *ping*, the file will be annotated and mailed to the e-mail

addressee. The same file will be copied to the /Up directory and the down will be replaced by the timestamp as follows

steel.stl.nps.navy.mil.23.Jun.1996.1400. After the file has been archived, the down file will be deleted from the monitoring directory. If the *ping* again yields negative results, the down file is merely annotated.

***b. down status does not exist***

If a down status does not exist again two actions can occur based on a good *ping* or a bad *ping*. If it is a good *ping*, nothing happens. If it is the first bad *ping*, the down file is opened and annotated with the time of the first bad *ping*. An e-mail is forwarded to the e-mail addressee with the same information.

## LIST OF REFERENCES

Alles, Anthony, *ATM Internetworking*, Cisco Systems, Inc., San Jose, California, May 1995. Available at <http://www.cisco.com/warp/public/614/12.html>

ATM Forum, *53 BYTES, The ATM Forum Newsletter*. Vol 4, No 2. July 1996. Mountain View Ca. Available at <http://www.atmforum.com/>

*ATTILA Network Analyzer overview*, MCNC, North Carolina, October, 1995. Available at <http://www.mcnc.org/HTML/ITD/ANT/Attila.html>

Bigelow, John, *Internetworking: Planning and Implementing a Wide Area Network (WAN) for K-12 Schools*, Master's Thesis, Naval Postgraduate School, Monterey, California, 1996.

Brenner, Steven & Aoki, Edwin, *Introduction to CGI/PERL*, New York, NY, M&T Books, 1996. Available at <http://www.mispress.com/introcgi/>

Cisco's Internetworking Terms and Acronyms, 1995. Available at <http://www.erl.noaa.gov/noc/cisco/data/doc/cintrnet/ita.htm>

Coden, M. H., *The Fiber Optic LAN Handbook*, Fourth Edition, Codenoll Technology Corporation, Yonkers N.Y., October, 1991.

Courtney, Dale, *Internetworking: NPS ATM LAN*, Master's Thesis, Naval Postgraduate School, Monterey, California, 1996. Available at <http://www.stl.nps.navy.mil/~iirg/courtney/>

Deering, Stephen, *Ipv6: The Next-Generation Internet Protocol*, ACM SIGCOMM Conference Tutorial, Stanford University, 1996.

Dennis, Ronald, *Internetworking: IP/ATM LAN Security*, Master's Thesis, Naval Postgraduate School, Monterey, California, 1996. Available at <http://www.stl.nps.navy.mil/~iirg/dennis/>

Erdogan, Ridvan, *Internetworking: Using Global ATM Networks for Live Multicast Audio/Video Distribution*, Master's Thesis, Naval Postgraduate School, Monterey, California, 1995. Available at <http://www.stl.nps.navy.mil/~iirg/erdogan/>

ForeRunner, *ASX-200 ATM Switch User Manual*, sec 5.3.1, Fore Systems, Inc., Warrendale, PA, 1995. Available at <http://www.fore.com/html/products/adpt/>

National Communications Systems, *Glossary of Telecommunication Terms*, FED-STD-1037A, 1986.

Interview between Arul Ananthanarayanan, Senior UNIX Systems and ATM Manager UCSC, and author, March. 7, 1996.

Interview between Milena Cochran, Network Administrator of the System Technology Lab (STL) NPS , and author, April 14, 1996.

Interview between Ray Mclean, UNIX Systems Manager MLML, and author, August.30, 1996.

Krivda, Cheryl D. *Analyzing ATM Adapter Performance The real-World Meaning of Benchmarks*. Efficient Networks, Inc., 1996. Available at <http://www.efficient.com/dox/EM.html>

Macedonia, M, and Brutzman, D., *MBone provides Audio and Video Across the Internet*, IEEE Computer, vol. 27 no. 4 pp. 30-36, April 1994. Available at <ftp://taurus.cs.nps.navy.mil/pub/i3la/mbone.html>

McKenzie, Alex, "RFC941, Addendum to the Network Service Definition Covering Network Layer Addressing," April, 1985. Available at <http://www.cis.ohio-state.edu/htbin/rfc/rfc941.html>

Partridge, Craig, *Gigabit Networking*, Addison-Wesley, New York, 1994.

Paxson, Vern, *An Introduction to Internet Measure and Modeling*, ACM SIGCOMM Conference Tutorial, Stanford University, 1996.

Schulze, M., and Farrell, C. *Network monitoring and visualization tools*. July, 1996. available at <ftp://coast.cs.purdue.edu/pub/aux/tools>

Stallings, William, *Data and Computer Communications*, Macmillan Publishing Co., Englewood Cliffs, NJ, 1991.

Tamer, Murat, *Internetworking: Multicast and Atm Network Prerequisites for Distance Learning*, Master's Thesis, Naval Postgraduate School, Monterey, California, 1996. Available at <http://www.stl.nps.navy.mil/~iirg/tamer/>

Tiddy, Michael, *Internetworking: Economical Storage and Retrieval of Digital Audio and Video for Distance Learning*, Master's Thesis, Naval Postgraduate School, Monterey, California, 1996. Available at <http://www.stl.nps.navy.mil/~iirg/tiddy/>

Triticom, *Award Winning Software Tools for LAN Management*, Eden Prairie, Maine.  
Available at [www.triticom.com](http://www.triticom.com)

Weinman, William E., *The CGI Book*, New Riders Publishing, 1995. Available at  
<http://www.bearnnet.com/wew/>



## INITIAL DISTRIBUTION LIST

	No. Copies
1. Defense Technical Information Center.....	2
8725 John J. Kingman Road., Ste 0944	
Ft. Belvoir, VA 22060-6218	
2. Dudley Knox Library.....	2
Naval Postgraduate School	
411 Dyer Rd.	
Monterey, California 93943-5101	
3. Don Brutzman, Code UW/Br .....	4
Naval Postgraduate School	
Monterey, California 93943-5101	
4. Rex Buddenberg, Code SM/Bu .....	2
Naval Postgraduate School	
Monterey, California 93943-5101	
5. Dr. Jim Eagle, Chair, Code UW .....	1
Naval Postgraduate School	
Monterey, California 93943-5101	
6. LCdr Evan Edwards.....	3
Naval Postgraduate School, 1882	
Monterey, California 93943-1882	
7. Dr. James C. Emery, Code 05.....	2
Naval Postgraduate School	
Monterey, California 93943-5101	
8. Dr. Reuben Harris, Chair, Code SM/Ha.....	1
Naval Postgraduate School	
Monterey, California 93943-5101	
9. Dr. Ted Lewis, Chair, Code CS.....	1
Naval Postgraduate School	
Monterey, California 93943-5101	

10. Dr. G.M. Lundy, Code CS/Ln ..... 1  
 Naval Postgraduate School  
 Monterey, California 93943-5101
  
11. Dr. Luqi, Code CS/Lq..... 1  
 Naval Postgraduate School  
 Monterey, California 93943-5101
  
12. Don McGregor, Code CC ..... 1  
 Naval Postgraduate School  
 Monterey, California 93943-5101
  
13. Dave Norman, Code 51 ..... 1  
 Director, W.R. Church Computer Center  
 Naval Postgraduate School  
 Monterey, California 93943-5101
  
14. Gary Porter, Code CC ..... 1  
 Naval Postgraduate School  
 Monterey, California 93943-5101
  
15. Dr. Maxine Reneker, Code 52 ..... 1  
 Director, Dudley Knox Library  
 Naval Postgraduate School  
 Monterey, California 93943-5101
  
16. Lou Stevens, Code CS/St ..... 1  
 Naval Postgraduate School  
 Monterey, California 93943-5101
  
17. Dr. Murali Tummala, Code EC/Tu ..... 1  
 Naval Postgraduate School  
 Monterey, California 93943-5101
  
18. Terry Williams, Code CC ..... 1  
 Naval Postgraduate School  
 Monterey, California 93943-5101



19. Dr. Geoffrey Xie, Code CS/Xi..... 1  
Naval Postgraduate School  
Monterey, California 93943-5101
20. CAPT George Zolla, USN, Ret. .... 1  
Dudley Knox Library  
Naval Postgraduate School  
Monterey, California 93943-5101
21. Dr. Michael J. Zyda, Code CS/Zk ..... 1  
Naval Postgraduate School  
Monterey, California 93943-5101
22. Arul K. Ananthanarayanan ..... 1  
University of California Santa Cruz  
225 Applied Sciences Bldg  
Santa Cruz, California 95064
23. Roland Baker ..... 1  
Santa Cruz County Office of Education  
Media and Technology Services  
809 Bay Avenue  
Capitola, California 95010
24. Lt Col Bill Barattino, Code JEEA . .... 1  
Advanced Communications Technology Department  
Center for Systems Engineering  
Defense Information Systems Agency (DISA)  
10701 Parkridge Blvd  
Reston, Virginia 22091
25. CAPT Ross D. Barker..... 1  
Senior Member, Smart Ship Program  
Office of the Chief of Naval Operations  
10701 Parkridge Blvd  
Reston, Virginia 22091

26. Steve Batsell ..... 1  
 Network Research Group Leader  
 MS 6367, Bldg 6012  
 Oak Ridge National Laboratory  
 Oak Ridge, Tennessee 37831
  
27. Dr. Carl R. Berman, Jr. .... 1  
 Office of the Provost  
 CSU Monterey Bay  
 100 Campus Center  
 Seaside, California 93955-8001
  
28. Bill Brutzman ..... 1  
 3 South Kingman Road  
 South Orange, New Jersey 07079
  
29. Erik Chaum..... 1  
 Code 2251, Building 1171  
 Naval Undersea Warfare Center Division Newport  
 1176 Howell Street  
 Newport, RI 02841-1708
  
30. Marke Clinger ..... 1  
 FORE Systems, Inc.  
 174 Thorn Hill Road  
 Warrendale, PA 15086
  
31. RADM O. Combs ..... 1  
 SPAWAR-05  
 2451 Crystal Drive  
 Arlington, VA 22202
  
32. Dr. Tom Defanti..... 1  
 Electronic Visualization Laboratory  
 University of Illinois at Chicago, MC 154  
 851 S. Morgan St., Room 1120  
 Chicago, Illinois 60607-7053

33. Captain Dave Gamble ..... 1  
PMWW152  
2451 Crystal Drive  
Arlington, VA 22202
34. Dr. Nancy Giberson ..... 1  
Santa Cruz County Office of Education  
809 Bay Avenue  
Capitola, California 95010
35. Mr. Paul Goss ..... 1  
SEA04I, Dir IRM  
Naval Sea Systems Command  
2531 Jefferson Davis Hwy.  
Arlington, VA 22242
36. Bruce Gritton ..... 1  
Monterey Bay Aquarium Research Institute (MBARI)  
P.O. Box 628  
Moss Landing, CA 95039-0628
37. R. Peter Haddad, M/S 2L-3 ..... 1  
Hewlett-Packard Laboratories  
1501 Page Mill Road  
Palo Alto, California 94304-1126
38. Dr. Harold Hawkins..... 1  
Office of Naval Research  
Cognitive & Neural S&T Division  
Program Officer  
Attn: ONR 341  
800 North Quincy Street, Ballston Tower One  
Arlington, Virginia 22217-5660
39. Dr. G. Ross Heath..... 1  
Executive Director  
Monterey Bay Aquarium Research Institute (MBARI)  
P.O. Box 628  
Moss Landing, CA 95039-0628

40. Mike Herbst ..... 1  
Far West Laboratory  
730 Harrison Street  
San Francisco, California 94107-1241
  
41. Captain Roger Hull ..... 1  
PMW171  
2451 Crystal Drive  
Arlington, VA 22202
  
42. Birt Johnson ..... 1  
Pacific Bell  
340 Pajaro Street, Room 131  
Salinas, California 93901
  
43. Tom Karwin ..... 1  
U.C. Santa Cruz  
P.O. Box 7600  
Santa Cruz, California 95061-7600
  
44. Robert Kochanski ..... 1  
NRaD  
Code 80  
53560 Hull St.  
San Diego, California 92152-5001
  
45. Dr. Marc Lipman ..... 1  
Mathematical, Computer, and Information Sciences Division  
Office of Naval Research  
Ballston Centre Tower One  
800 North Quincy Street  
Arlington, Virginia 22217-5660
  
46. Dr. William J. "Bill" Lennon ..... 1  
Lawrence Livermore National Laboratory  
7000 East Avenue (L-541)  
Livermore, California 94550-9900

47. Syd Leung ..... 1  
Pacific Bell  
2600 Camino Ramon, Room 35306  
San Ramon, California 94105
  
48. Jeff Liening ..... 1  
FC4A/TNAC - Validation Support Branch  
203 West Losey Street, Room 2000  
Scott AFB, IL 62225-5238
  
49. Jack Linthicum ..... 1  
Technical Information Specialist  
New Technology Development Division  
Office of Engineering and Technology  
Federal Communications Commission  
1919 M Street N.W.,  
Washington DC 20554
  
50. Brian Lloyd ..... 1  
Lloyd Internetworking  
3461 Robin Lane  
Cameron Park, California 95681
  
51. Cdr. Mike Loescher ..... 1  
Special Asstistant for IW  
Deputy Assistant Secretary of the Navy (DASN)  
Washington, D.C. 20350-1000
  
52. Dr. Melanie Loots ..... 1  
Associate Director, National Center for Supercomputing Applications (NCSA)  
4119 Beckman Institute, MC 251  
University of Illinois at Urbana-Champaign  
605 East Springfield Ave.  
Champaign, Illinois 61820
  
53. Dr. Michael R. Macedonia ..... 1  
Fraunhofer Center for Research in Computer Graphics, Inc.  
Vice-President for Developing Global Work Environments  
167 Angell Street  
Providence, RI 02906

54. Rick Maciel..... 1  
Pacific Bell ATM Project Manager  
2600 Camino Ramon, Room 4S155  
San Ramon, California 94583
55. Lora Lee Martin, Director ..... 1  
Director of Program and Policy Development  
University of California, Santa Cruz  
Santa Cruz, California 95064
56. Kam Matray ..... 1  
Monterey Bay Technology Education Center,  
Monterey Peninsula Unified School District  
PO Box 1031  
Monterey, California 93942-1031
57. Chris May ..... 1  
California State University, Monterey Bay  
100 Campus Center  
Seaside, California 93955-8001
58. Dr. James H. May ..... 1  
California State University, Monterey Bay  
100 Campus Center  
Seaside, California 93955-8001
59. Michael McCann ..... 1  
Monterey Bay Aquarium Research Institute  
P.O. Box 628  
Moss Landing, California 95039-0628
60. Dr. Ray McClain..... 1  
Moss Landing Marine Laboratories  
P.O. Box 450  
Moss Landing, California 95039
61. Mr. Jim McDonnell ..... 1  
SEA03KM  
Naval Sea Systems Command  
2531 Jefferson Davis Hwy.  
Arlington, VA 22242

62. Mike Mellon ..... 1  
Monterey County Office of Education  
Instructional Resources and Technology  
PO Box 80851  
Salinas, California 93912-0851
  
63. Dr. Pat Mantey ..... 1  
Chair, Computer Engineering  
University of California, Santa Cruz  
Santa Cruz, California 95064
  
64. Captain Mark Moranville ..... 1  
SEA03J, Dir Integrated IS Group  
Naval Sea Systems Command  
2531 Jefferson Davis Hwy.  
Arlington, VA 22242
  
65. Michael Newman ..... 1  
Newman and Associates  
24090 Summitwoods Drive  
Los Gatos, California 95030
  
66. RADM(Sel) Kate Paige ..... 1  
Commander NSWC  
10901 New Hampshire Ave  
Silver Spring, MD 20903-5640
  
67. Richard Pearlman ..... 1  
Technical Director, Knowledge Network  
2150 Webster, Room 440  
Oakland, California 94611
  
68. Capt Jim Quiros ..... 1  
FC4A/TNAC - Validation Support Branch  
203 West Losey Street, Room 2000  
Scott AFB, IL 62225-5238
  
69. Captain Charles Ristorcelli, PD16 ..... 1  
2451 Crystal Drive  
Arlington, VA 22202

70. Emily Routman ..... 1  
 Exhibit Developer, San Jose Tech Museum of Innovation  
 145 West San Carlos Street  
 San Jose, California 95113
  
71. Kathy Ruthowski ..... 1  
 Editor, NetTeach News  
 13102 Weather Vane Way  
 Herndon, Virginia 22071-2944
  
72. Dr. John Schill ..... 1  
 Program Manager  
 ISO Planning and C3  
 Defense Advanced Research Projects Agency  
 3701 N. Fairfax Drive  
 Arlington, Virginia 22203-1714
  
73. Dr. Arthur St.George ..... 1  
 National Science Foundation  
 4201 Wilson Boulevard  
 Arlington, Virginia 22230
  
74. Dr. Fred Siff, Associate Vice Chancellor ..... 1  
 Communications and Technology Services  
 University of California, Santa Cruz  
 1156 High Street  
 Santa Cruz, California 95064
  
75. G. (Joe) Simone ..... 1  
 Executive Director  
 FasTrak Advanced Digital Services  
 2600 Camino Ramon, Room 4S155  
 San Ramon, California 94583
  
76. Diane Siri ..... 1  
 County Superintendent of Schools  
 Santa Cruz County Office of Education  
 809 Bay Avenue  
 Capitola, California 95010



77. Captain Ken Slaght, PMW176 ..... 1  
2451 Crystal Drive  
Arlington, VA 22202
  
78. Dr. Larry Smarr..... 1  
Director, National Center for Supercomputing Applications (NCSA)  
155 Computing Applications Building, MC 476  
University of Illinois at Urbana-Champaign  
605 East Springfield  
Champaign, Illinois 61820
  
79. Bradley R. Smith, Computer Facilities Manager..... 1  
University of California, Santa Cruz  
145 Applied Sciences Bldg  
Santa Cruz, California 95064
  
80. David Sonderegger ..... 1  
Pacific Bell ATM Project Manager  
2600 Camino Ramon, Room 4S155  
San Ramon, California 94583
  
81. LCDR Brian Steckler, USN, Ret. .... 1  
25381 Carmel Knolls Drive  
Carmel, California 93923
  
82. David Stihler..... 1  
Monterey County Information Systems  
1590 Moffett Street  
Salinas, California 93905-3341
  
83. Chris Taylor ..... 1  
Director of Computing and Computer Resources (CCR)  
California State University, Monterey Bay  
100 Campus Center  
Seaside, California 93955-8001
  
84. 2Lt Fred Taylor..... 1  
FC4A/TNAC - Validation Support Branch  
203 West Losey Street, Room 2000  
Scott AFB, IL 62225-5238

85. Dr. Anujan Varma ..... 1  
Associate Professor of Computer Engineering  
University of California, Santa Cruz  
1156 High Street  
Santa Cruz, California 95064
86. Jim Warner ..... 1  
Network Engineer, University of California Santa Cruz  
C A T S/Network & Telco Services  
11 Communications Bldg  
Santa Cruz, California 95064
87. David Warren ..... 1  
Cabrillo College  
6500 Soquel Drive  
Aptos, California 95003
88. Steve Watkins ..... 1  
University of California Santa Cruz  
Science Library  
Santa Cruz, California 95064
89. Dr. Glen H. Wheless ..... 1  
Center for Coastal Physical Oceanography  
Old Dominion University  
Norfolk, Virginia 23529